Open Research Online

The Open University's repository of research publications
and other research outputs

Full report on challenges with learning to program and problem solve: an analysis of first year undergraduate Open University distance learning students' online discussions

Other

How to cite:

Savage, Simon and Piwek, Paul (2019). Full report on challenges with learning to program and problem solve: an analysis of first year undergraduate Open University distance learning students' online discussions. The Open University, Milton Keynes.

For guidance on citations see FAQs.

© 2019, 2020 The Authors

https://creativecommons.org/licenses/by-nc-nd/4.0/

Version: Version of Record

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data policy on reuse of materials please consult the policies page.

oro.open.ac.uk

Full report on challenges with learning to program and problem solve: an analysis of first year undergraduate Open University distance learning students' online discussions

SIMON SAVAGE* and PAUL PIWEK*, School of Computing and Communications, The Open University, United Kingdom

Students who study problem solving and programming (in a language such as Python) at University level encounter a range of challenges, from low-level issues with code that won't compile to misconceptions about the threshold concepts and skills. The current study complements existing findings on errors, misconceptions, difficulties and challenges obtained from students through after-the-fact instruments such as questionnaires and interviews. In our study, we analysed the posts from students of a large cohort (~1500) of first-year University distance learning students to an online 'Python help forum', recording issues and discussions as the students encountered specific challenges. Posts were coded in terms of topics, and subsequently thematically grouped into python-related, problem-solving/generic, and module-specific topics. This report documents the full set of topics and the statistics for each of them. We also provide examples from the forum discussions which illustrate the topics that were identified.

CCS Concepts: • **Social and professional topics** → **Computing education**; *Computational thinking*; *Software engineering education*; *Adult education*; • **Applied computing** → **Distance learning**; **E-learning**.

Additional Key Words and Phrases: programming, Python, problem solving, online student discussions, challenges, misconceptions, threshold concepts and skills

*This work has been completed with support from The Institute of Coding, an initiative funded by the UK Office for Students.

Authors' address: Simon Savage, s.a.savage@open.ac.uk; Paul Piwek, paul.piwek@open.ac.uk, School of Computing and Communications, The Open University, United Kingdom, Walton Hall, Milton Keynes, Buckinghamshire, MK7 6AA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice.

© 2019, 2020 The authors

Contents

List of Tables

List of Figures

1 INTRODUCTION

The introduction to a text-based procedural programming language poses a range of challenges for students, especially when this is done in a distance learning environment. Introducing students to programming (and problem solving) is generally acknowledged as difficult or at least perceived as a significant challenge [4, 5].

Students encounter threshold concepts and skills [9, 15, 16] and can get entangled in misconceptions [3, 13]. These issues may be amplified in distance and, more specifically, online learning environments, which have gained ground not only in the form of MOOCs and courses offered by online education providers, but also in more traditional teaching establishments that apply the 'flipped classroom' approach [1].

The purpose of the current study is to examine the challenges that first year distance learning students face when learning a procedural programming language such as Python over the course of a 21-week computing and IT course which includes 6 weeks on problem solving and programming with Python. So far, much of the evidence on students' challenges is based on retrospective surveys/interviews with students and instructors, and analysis of student assessment results or student-authored concept maps [13, 16]. To our knowledge, no results draw on large data sets of contemporary student discussions in their own language around challenges. The current study provides an additional perspective by examining the evidence from a large collection of student online discussions as they learn to program.

In Section 2, we summarise research methods and findings from the literature on the challenges faced by students when learning to program. Section 3 introduces the methodology of the current study which uses (online) discussions that took place as students encountered and struggled with specific challenges. Section 4 presents the results of our study, grouping challenges by topic and, at a higher level, into themes. This sections gives statistics for each topic that was identified. Section 5 provides specific examples from the forum discussions. This report concludes with a discussion of our findings and recommendations based on our findings (Section 6).

This report is intended as a companion to [11]. Sections 2 and 3 of this report overlap with [11] and can be skipped by readers already familiar with the paper.

2 RELATED WORK

A wide range of instruments have been used to study the challenges that students face when learning to program.

Perhaps the most direct evidence comes from studies that analyse compiler error messages when students attempt to execute their code. [8] found that such errors (in Java) be classified into different types with good reliability. Errors found this way can be ordered by frequency

and therefore provide some indication of their prevalence. However, the error type are relatively low-level (e.g. the top five reported by [8]) are: variable not declared, colon missing, incorrectly written variable name, invalid syntax, method naming incorrect.

Using the terminology of [13] most of these errors go back to difficulties in syntactic knowledge. However, [13] identify two further levels at which misconceptions can arise: the conceptual and strategic levels. E.g., at the conceptual level a student may fail to understand that a variable can only hold one value at a time. Strategic knowledge or know how concerns expertise on how to go from a problem to an implemented solution for that problem. This includes for instance problem decomposition, development of an algorithm and implementation of this algorithm in a specific language, whilst testing and debugging the code as it is developed.

Within computing, specific attention has been given to 'Threshold concepts', originally proposed by [9] as transformative, integrative, irreversible, potentially troublesome and often indicating the boundaries of a discipline. Early research identified for instance objected orientation and pointers as threshold concepts [2]. More recently, [16], in their literature review, enumerate a long list of further concepts that have since been identified, from data abstraction and design patterns to polymorphism and program-memory interaction. Additionally, [15] highlight that not only concepts, but also skills, can play the role of thresholds in computing.

[16] enumerate the instruments that have been used so far for collecting data on threshold concepts. They distinguish evidence from faculty and students. Much of the evidence focuses on after the fact data collection through surveys and interviews. Similarly, work building concept inventories (e.g. [3], a concept inventory is multiple-choice questionnaire where each distractor answer maps to a specific student misconception) typically relies on questionnaires to gather initial evidence on where students face difficulties in understanding concepts.

[18] argue that collecting evidence from students on where they experienced difficulties in the past can be unreliable, with students struggling to recall and recount specific occasions on which challenges were encountered. This suggest that records of challenges as they occur may be more informative on actual problems that students encounter.

As described by [10], specific challenges can arise in distance learning contexts. These are related to the type of students (distance learning students often study part-time whilst being in full-time employment) and reliance on technology to allows students to communicate and collaborate with each other.

3 METHODOLOGY

Our research question is: *What are the challenges that first-year distance learning students face when learning to program in Python?* Rather than rely on post-hoc accounts of what students remember

about the challenges they encountered, we examine records of discussions around the challenges *as they emerged* for our students.

The course in question, *Introduction to computing and information technology 2* (henceforth TM112), included a dedicated online 'Python help forum' where students could discuss challenges with their study of Python. The online forum relied primarily on peer support (enabled partly by a wide mix of abilities among our students, from absolute novices to programming experts seeking a qualification). We also observed that students were particularly effective in motivating and providing emotional support to each other, where they faced difficulties. In addition to the peer support, there was a team of moderators (three) who monitored the forum discussions and would steer the discussion if needed.

3.1 Student demographics

We studied the 2018 April-September TM112 cohort of about 1500 students. More than half of students were between 25 and 39 years old; 10 students were over 65 years old; approximately 200 were under 21. The female to male ratio is 24 to 76. Race/ethnicity of students (rounded): 89% of students were White, 2% Black, 3% Asian, 1% Mixed, 1% Other and 3% did not specify. 19% of students declared a disability. 16% of students are classified as low socio-economic status by the university, with a further 5% being unknown.

TM112 is the second module for most Computing and Information Technology undergraduate students at the University and also taken by some students on other pathways, such as Data Science. Most students will have completed a predecessor module *Introduction to computing and information technology 1* (TM111, see [19]) which introduces students to University level study, a range of computing and IT topics and basic programming in a visual programming language.

All students (both those who completed and those who did not complete the module) are surveyed at the end of the module (before they receive their results). The response rate for the survey for the April 2018 cohort was 16%. Most students, over 90% agreed with the statement 'I was satisfied with the quality of the module' (and less than 5% disagreed). About 80% agreed that 'My studies have helped me develop my self-confidence', (less than 4% disagreed). Also, more than 90% agreed that 'It was obvious how the module materials related to the assessed tasks on this module', (less than 3% disagreed).

3.2 Instructor demographics

The online 'Python help' forum was moderated by 2 tutors (male and female). Additionally, the course leader (male) monitored the forums and supported the two tutors. All three staff involved are white and have, each individually, over 20-years of computing and Information Technology teaching experience.

3.3 Programme components

6 of the 21 study weeks of TM112 are on 'Problem solving with Python':

Week 2 Sequence, selection, variables, lists and (nested) iteration, Python Turtles library.

Week 4 Formula problems, case analysis, Booleans, testing, documentation, pattern for generating a sequence.

Week 7 Generating lists, Reduce (count and aggregate), Search (finding a value/the best value), Combining patterns.

Week 9 Python functions (and automated testing of functions with assert), Python objects and names.

Week 10 Worked example of analysis, with Python, of health and wellbeing dataset.

Week 15 Worked example using Python dicts, interactive loops, the random library and reading from a file.

[12] provides more information about the approach taken in TM112.

The results in this study concern the cohort that took TM112 from April to September 2018. During each week, students work with printed materials and online activities and during some weeks attend tutorial events (face-to-face or online). At the end of each week, there is a formative quiz. To encourage students to engage with the quizzes, they are rewarded with a small number of marks for including evidence of engagement with the questions in their assignments. Marks are for the evidence of engagement and personal narrative/reflection on their engagement with the quiz questions. Since the quiz questions were not summatively assessed, students were also encouraged to discuss their attempts and answers with their peers on the module forums. For the weeks covering Python, see above, they are referred specifically to the Python help forum.

Students were made aware of the forums through several routes. Initial contact from their tutors mentioned the forums and that it is permissible to discuss quiz questions in them. This latter point was reinforced in the introductory video that students watch during their first week of study. Students were also encouraged to explore the module website, including the forums, during that first week and the 'Python Help' contained a pinned post explaining that it is okay to discuss quiz questions in it. Students who posted in less appropriate forums were redirected to the correct forum. Forums consist of a series of discussions as illustrated in Figure 1 (personal information has been hidden).

Each discussion consists of one or more posts as shown in Figure 2

3.4 Data analysis methods

Our raw data was the full collection of posts to the Python help forum. We extracted the post data from the forum site and stored it in a spreadsheet – see [17] for the scripts used to extract the

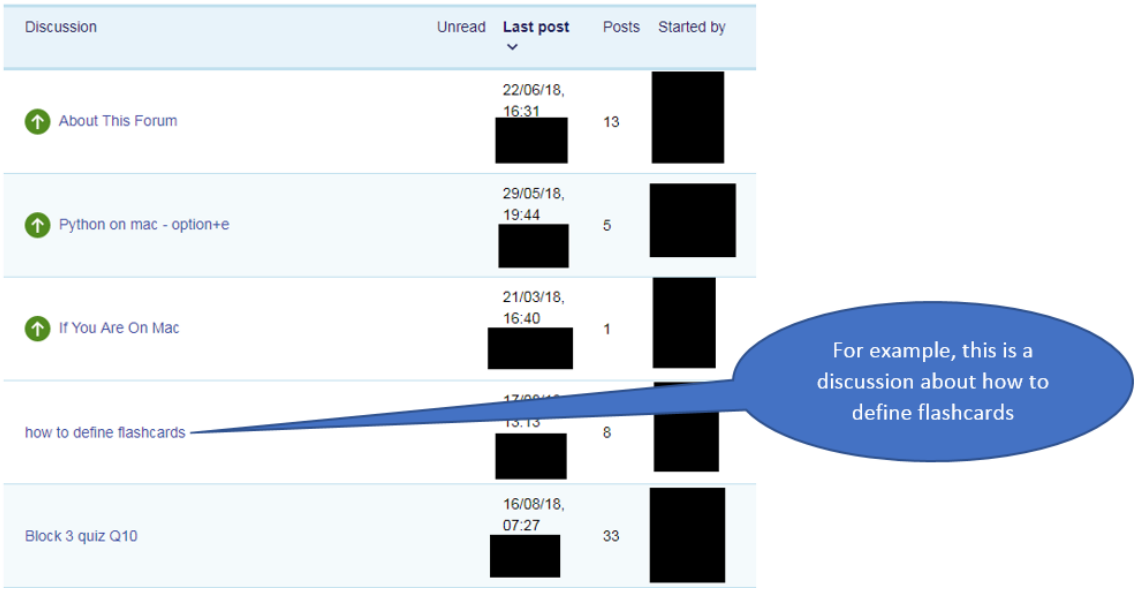


Fig. 1. Forum consisting of a series of discussions

data. Information which could identify participants was anonymised. One or more topics were assigned to each post, providing a descriptive label for the content of the post (i.e. during the First Cycle coding we applied Descriptive coding [14]). In the first instance, all coding was carried out by one of the authors and subsequently validated by the other author. On some occasions, some topics were combined. A second thematic grouping of the topic labels was carried out (as part of the Second Cycle coding [14]) [1]. This led us to a division of topics into three higher level themes: python-related, problem-solving/generic programming-related, and module-specific. Based on the qualitative coding, certain quantitative information emerged which is presented in the results section.

3.5 Scope and limitations

There is a limitation with the core data in that it doesn't record the engagement of non-speakers. This means that we can't accurately measure how individual discussions reverberated within the student community. The forum software does track the most read discussions with the top five being read 250, 268, 277, 311 and 386 times respectively. Thus we can be confident that discussions are being read with some attracting hundreds of readers.

[1] Additional annotation of emotions and the nature of turns was also recorded. We may analyse these at a later date.

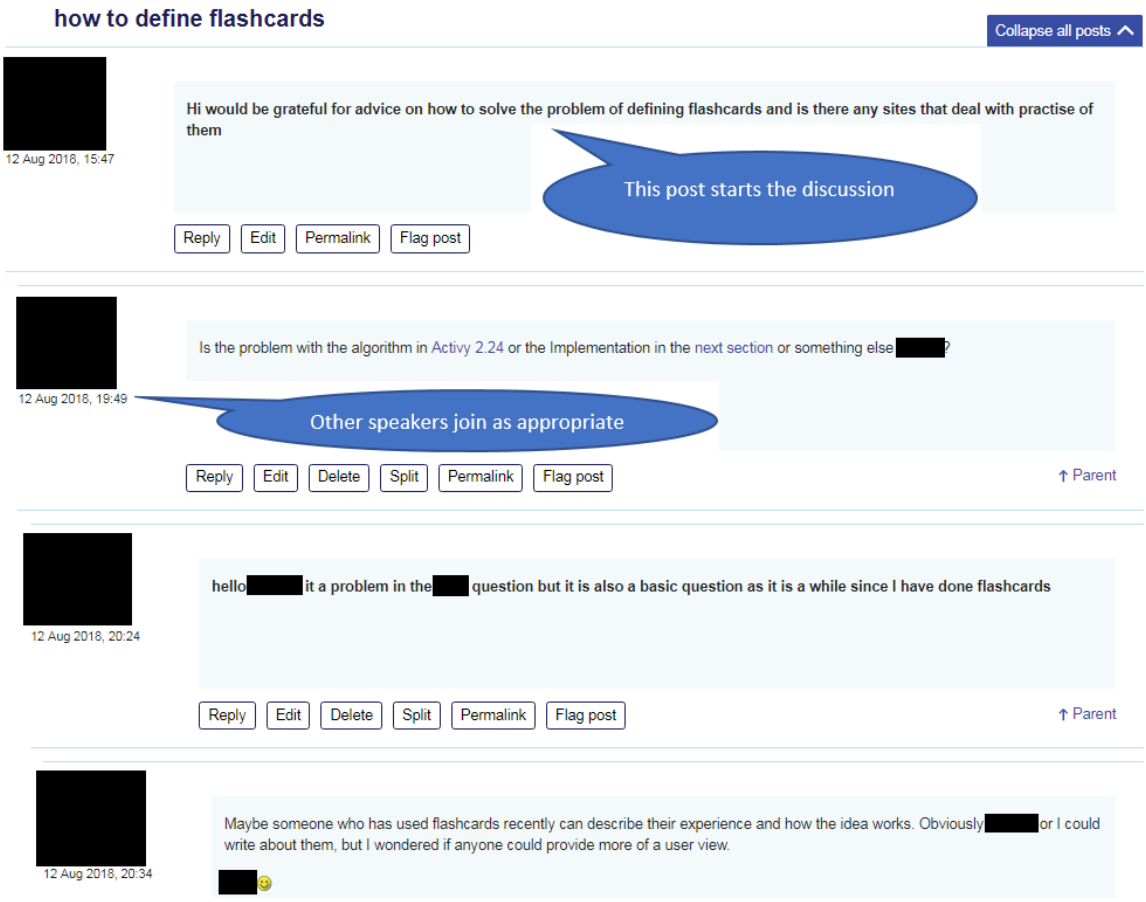


Fig. 2. Each discussion consists of a series of posts

The methodology does have other potential limitations. The manual nature of the coding process means that it may be biased by the opinion of the encoder's choice of a single topic and turn-type per turn although this is somewhat mitigated by team members reviewing the coding process.

Furthermore, the 'Python Help' forum's prime purpose was to support students in their understanding of Python concepts and problem-solving concepts raised in TM112 (see the list in Section 3.3 on Page 10). Topics which are beyond TM112 are unlikely to be raised, thus skewing the results in favour of TM112-related concepts.

Similarly, the student cohort in question was constituted primarily of part-time distance learning undergraduate computing and information technology students, whose profile is likely to differ from, for instance, computer science students. For many of our students, TM112 helps to decide the subsequent study pathway, with options ranging from computer science (with a significant programming and theoretical CS component) to information and communications technology.

The "Python Help" forum is unlikely to be students' only point of support and we could reasonably expect them to utilise their tutors; external social network facilities; and specialist websites for support. It is not possible to quantify the effect of this on our results.

All data collection and analysis complied with approval processes and methods at the University, ensuring participant privacy, confidentiality, and protection. Neither participants nor researchers received monetary or gift incentives. The data analysis was financially supported through a grant from *The Institute of Coding*.

4 RESULTS: TOPIC STATISTICS

Our overall aim was to learn which challenges students face on their first encounter (during their studies) with a text-based procedural programming language.

We tagged student posts on the module forums with topics. This gives us some indication of the specific topics that pose challenges. Additionally, we counted the number of discussions in which each topic was mentioned as a rough estimate of their prevalence as a challenge for members of the current student cohort. The topics were also categorised as to whether they belonged to one of three themes:

- *Python-related*,
- *problem solving/generic programming-related*, or
- *module-specific*.

The Python help forum contained 178 discussions with a total of 1430 posts. Nine discussions containing 29 posts were present which didn't fall within the scope of this study, such as one informing students of upcoming tutorials. These discussions have been maintained to ensure completeness of data but given the post type and topic type of 'irrelevant'. Thus, there are 169 relevant discussions containing 1401 posts. The encoding process identified 66 topics in the forum posts:

- 30 Python-related (see Section 4.2 from Page 19),
- 21 on problem solving and general programming skills (see Section 4.3 from Page 22), and
- 15 focusing on module-specific questions/issues (see Section 4.4 from Page 25).

Since topics are allocated to posts, each discussion may represent several topics. Tables 2 (Page 19), 3 (Page 22) and 4 (Page 25) show the number of discussions per topic relative to these categories.

163 people actively engaged (posted) in the forum, of these 154 were students. The remaining 9 were moderators, module team members and tutors. 105 students posted 4 times or fewer indicating that 49 students were responsible for most of the engagement. Many members of the TM112 cohort did not post in the forum although a substantial number regularly read them.

Topics were also analysed for an indication of engagement in terms of the number of discussions they appeared in; the minimum and maximum number of posts in those discussions; and the mean and median number of posts in those discussions. Tables 2, 3 and 4 also show these statistics.

4.1 Statistics relating to all topics

In this section we provide some statistics on the topics, without grouping them by theme. However, before we look at the topics themselves, we want to highlight that the student postings to the forum discussion consists of a wide variety of contribution types or dialogue acts. For information, we show the types we identified in Table 4.1.

Table 1. List of post types

Acknowledgment	Acknowledgment/thank you	Admonishment
Agree	Celebrating	Clarification
Complaint	Contradiction/Explanation	Demonstration
Direction to module material	Direction to other forum	Direction to other sources
Direction to other threads	Direction to tutor	Elaboration
Explanation	Explanation/admonishment	Explanation/question
External source	General conversation	Giving up
Irrelevant	No message	Question
Question/request for help	Reassurance	Reassurance/Suggestion
Recommendation	Reflection	Request for elaboration
Request for explanation	Self-answered	Sharing resource
Shutting down conversation	Statement of current understanding	
Suggestion	Walkthrough	

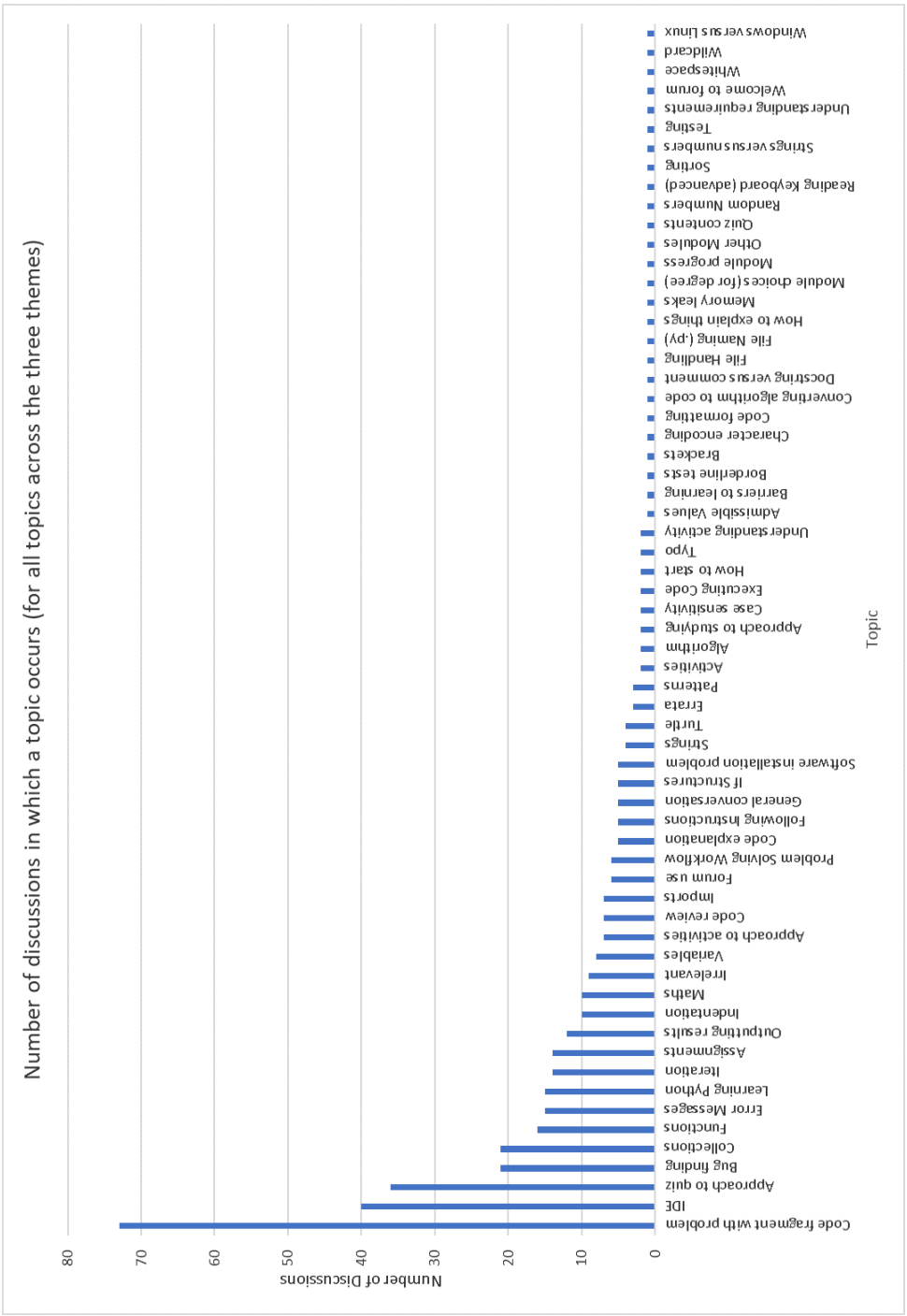


Fig. 3. Number of discussions in which a topic occurs (for all topics across the three themes)

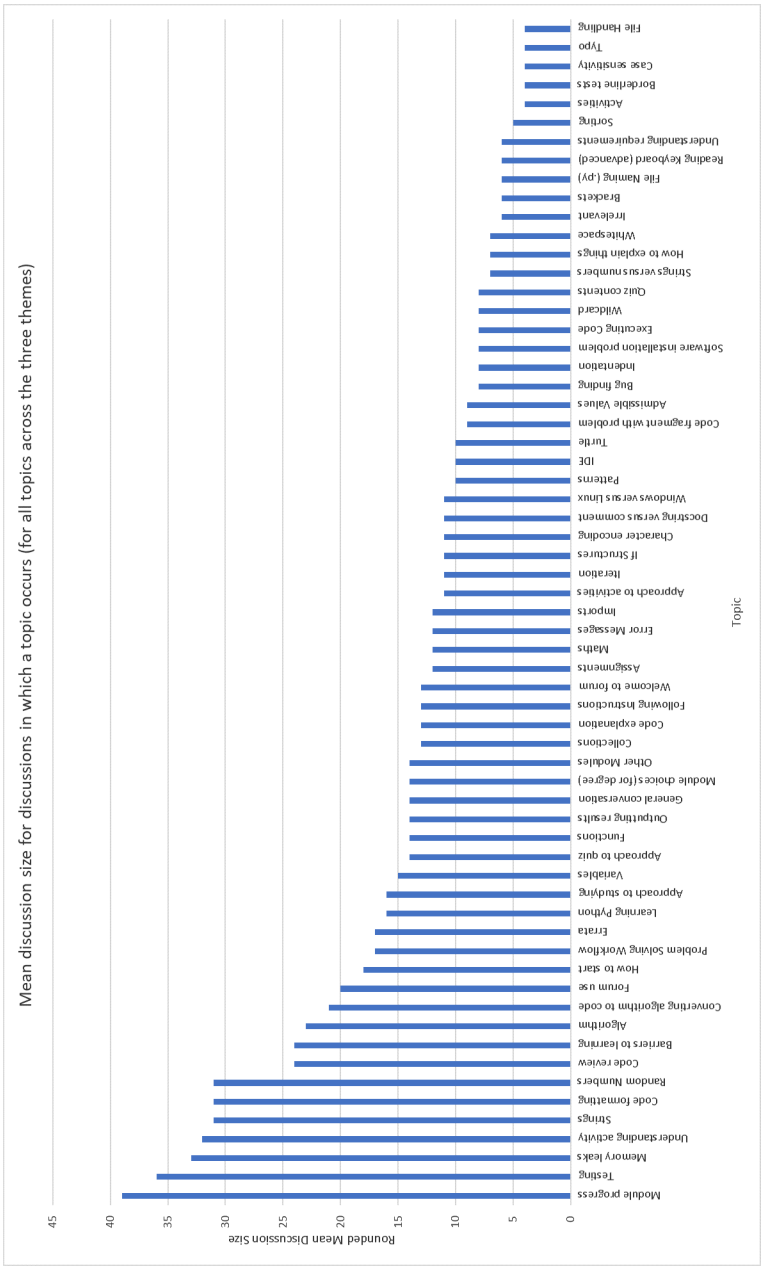


Fig. 4. Mean discussion size for discussions in which a topic occurs (for all topics across the three themes)

4.2 Python-related topics

Table 2. Discussion statistics by Python-related topic

Topic	Number of discussions	Min. discus. size (posts)	Max. discus. size (posts)	Rounded Mean discus. size (posts)	Rounded Median discus. size (posts)
IDE	40	1	57	10	7
Collections	21	4	57	13	8
Functions	16	2	31	14	12
Error messages	15	1	57	12	10
Iteration	14	2	39	11	9
Outputting results	12	2	33	14	9
Indentation	10	3	16	8	6
Variables	8	3	57	15	8
Imports	7	7	17	12	11
If-Structures	5	2	30	11	5
Software installation problem	5	1	17	8	5
Strings	4	12	57	31	27
Turtle	4	6	17	10	9
Case sensitivity	2	4	4	4	4
Executing code	2	8	9	8	9
Typo	2	4	5	4	5
Brackets	1	6	6	6	6
Character encoding	1	11	11	11	11
Code formatting	1	31	31	31	31
Docstring versus comment	1	11	11	11	11
File handling	1	4	4	4	4
File Naming (.py)	1	6	6	6	6
Memory leaks	1	33	33	33	33
Random numbers	1	31	31	31	31
Reading keyboard	1	6	6	6	6
Strings versus numbers	1	7	7	7	7
Whitespace	1	7	7	7	7
Wildcard	1	8	8	8	8
Windows versus Linux	1	11	11	11	11

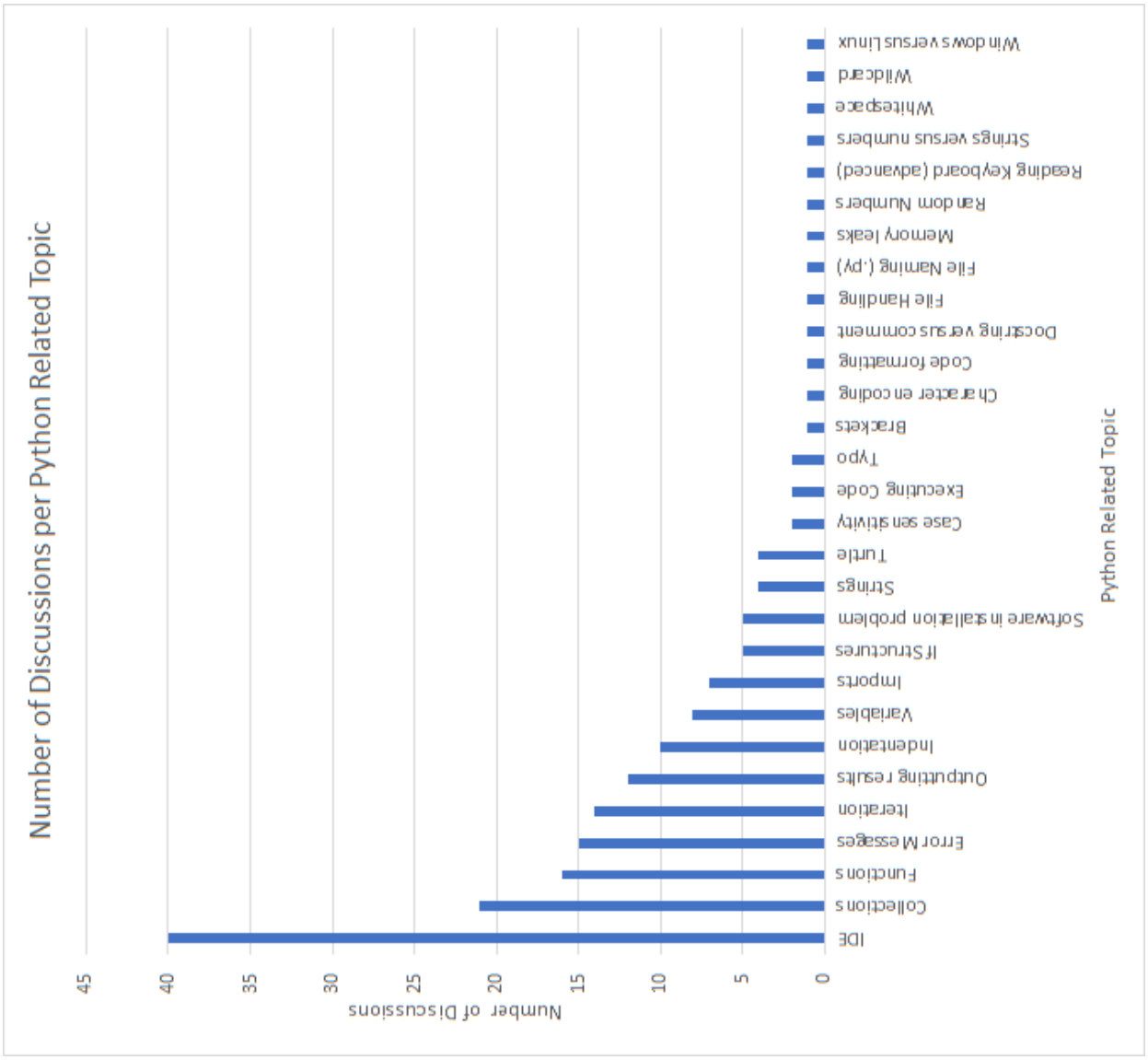


Fig. 5. Number of discussions per Python-related topic

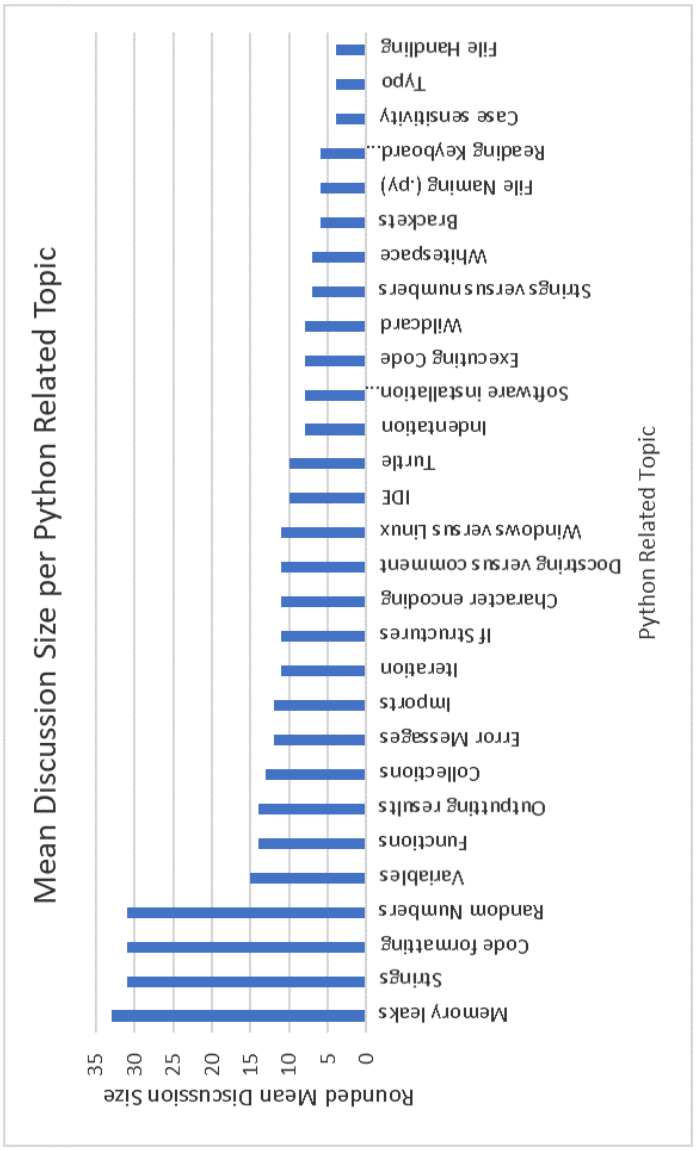


Fig. 6. Rounded mean discussion size (measured in number of posts) per Python-related topic

4.3 Problem-solving/generic topics

Table 3. Discussion statistics for each problem-solving/generic topic

Topic	Number of discussions	Min. discus. size (posts)	Max. discus. size (posts)	Rounded Mean discus. size (posts)	Rounded Median discus. size (posts)
Code fragment with problem	73	1	57	9	6
Bug finding	21	2	20	8	5
Learning Python	15	6	57	16	14
Maths	10	2	57	12	7
Code review	7	7	57	24	21
Problem-solving Workflow	6	1	30	17	17
Code explanation	5	5	30	13	8
Following Instructions	5	10	18	13	12
Patterns	3	7	12	10	10
Algorithm	2	10	36	23	23
How to start	2	7	30	18	19
Admissible Values	1	9	9	9	9
Barriers to learning	1	24	24	24	24
Borderline tests	1	4	4	4	4
Converting algorithm to code	1	21	21	21	21
How to explain things	1	7	7	7	7
Sorting	1	5	5	5	5
Testing	1	36	36	36	36
Understanding requirements	1	6	6	6	6

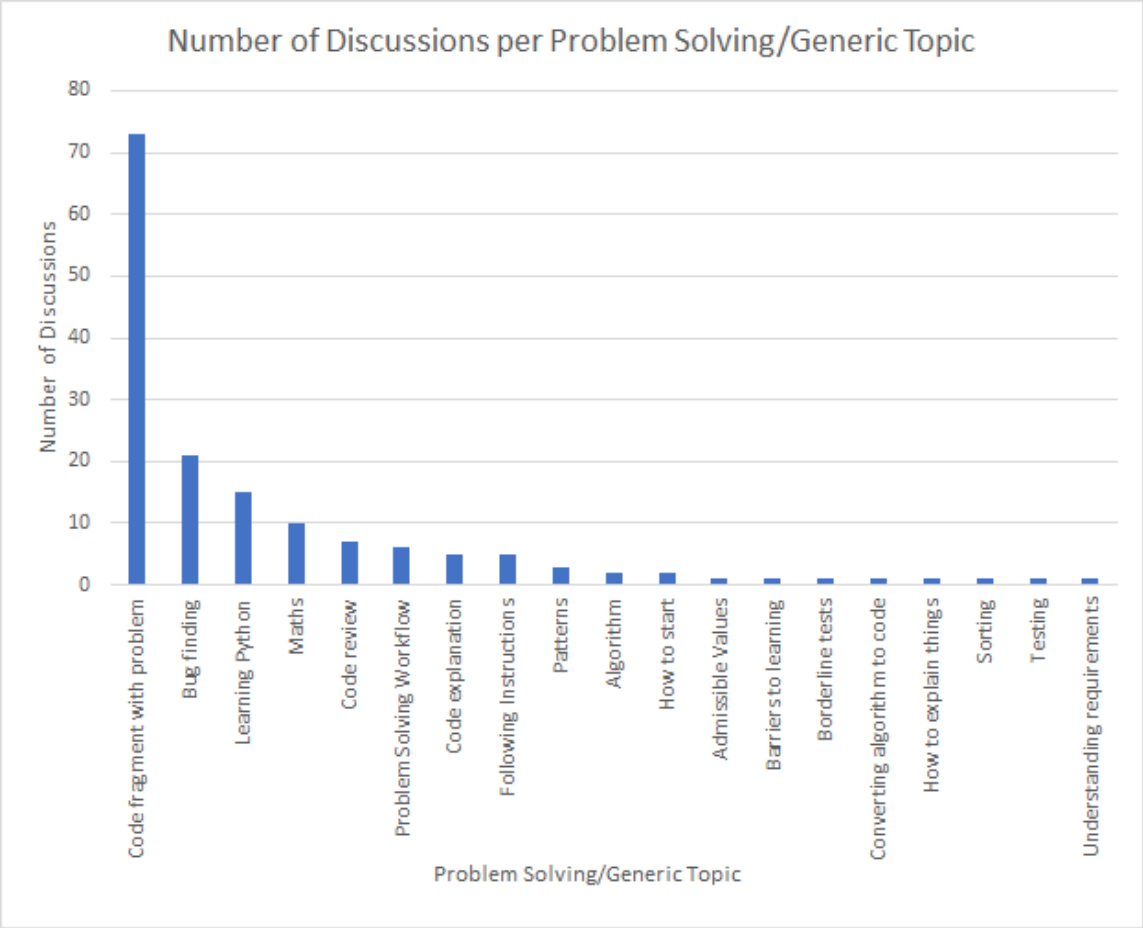


Fig. 7. Number of discussions per problem-solving/generic topic

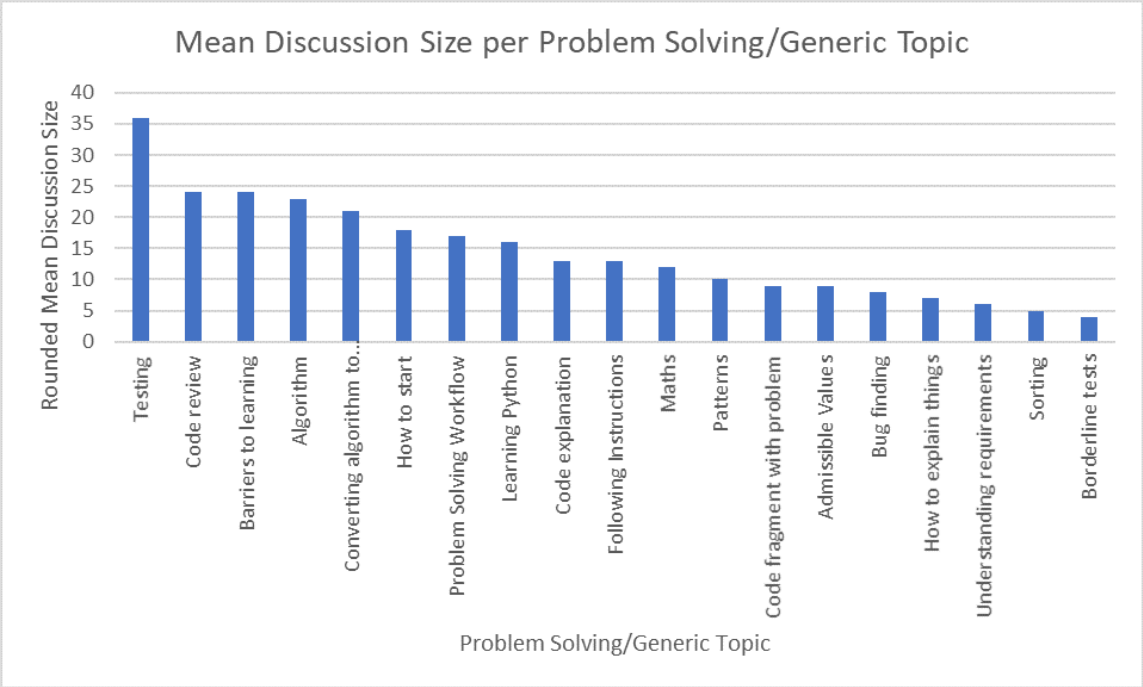


Fig. 8. Rounded mean discussion size (measured in number of posts) per problem-solving/generic topic

4.4 Module-specific topics

Table 4. Discussion statistics for each module-specific topic

Topic	Number of discussions	Min. discus. size (posts)	Max. discus. size (posts)	Rounded Mean discus. size (posts)	Rounded Median discus. size (posts)
Approach to quiz	36	1	57	14	11
Tutor Marked Assignment	14	3	36	12	10
Irrelevant	9	1	24	6	4
Approach to activities	7	3	39	11	4
Forum use	6	4	57	20	12
General conversation	5	6	33	14	9
Errata	3	3	39	17	10
Activities	2	2	7	4	5
Approach to studying	2	4	28	16	16
Understanding activity	2	8	57	32	33
Module choices (for degree)	1	14	14	14	14
Module progress	1	39	39	39	39
Other Modules	1	14	14	14	14
Quiz contents	1	8	8	8	8
Welcome to forum	1	13	13	13	13

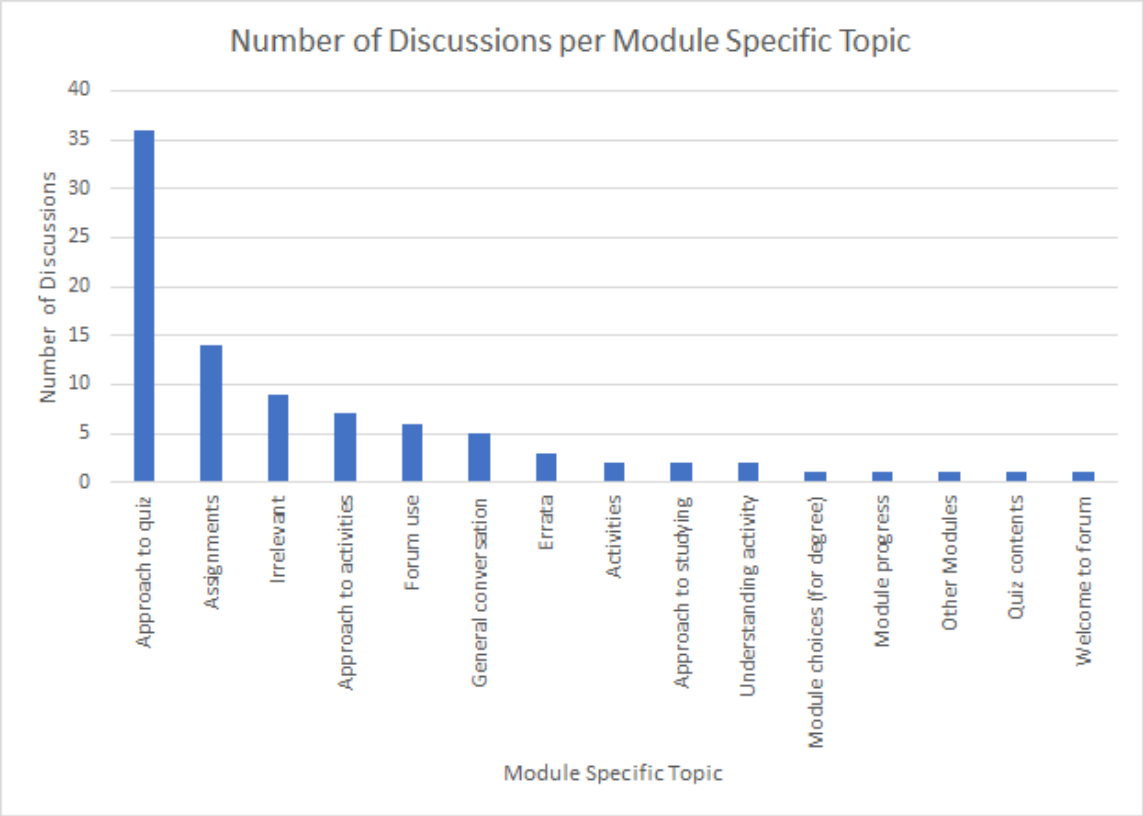


Fig. 9. Number of discussions per module-specific topic

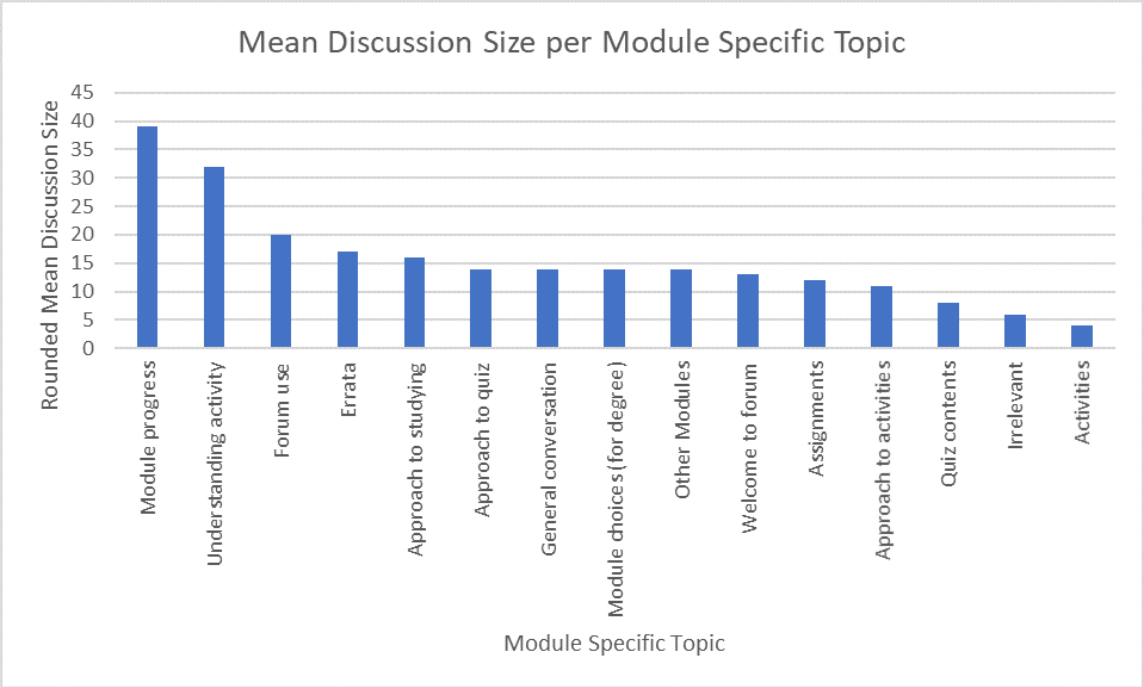


Fig. 10. Rounded mean discussion size (measured in number of posts) per module-specific topic

5 RESULTS: TOPIC DESCRIPTIONS AND EXAMPLES

In this section, we present examples of postings for the topics together with a supporting narrative. Where there were further subdivisions within a specific topic, subtopics are identified using the format: 'topic : subtopic' in the section headings. Examples of all topics are given but narration is reduced for less frequent topics (those which appeared in 5 or less discussions), since there was too little data to generalise specific findings for these topics.

5.1 Python-related topics

Python-related topics are those whose details are uniquely or significantly peculiar to Python in the context of the module. Similar topics may exist for other languages but the specifics will vary dramatically. For example configuring the Python IDE will be different to that for, say, Java or C++.

5.1.1 IDE: Shell vs Editor. The IDE was a topic in 40 discussions (24%).

Prominent in these was confusion between how to use the IDLE Shell and the Editor.[2] For example, a student experienced a syntax error with the following code as shown in Figure 11. It

[2]IDLE is the standard/default IDE for Python.

transpires that they had created the code in the shell, saved it as a '.py' file and then tried opening it in the editor.

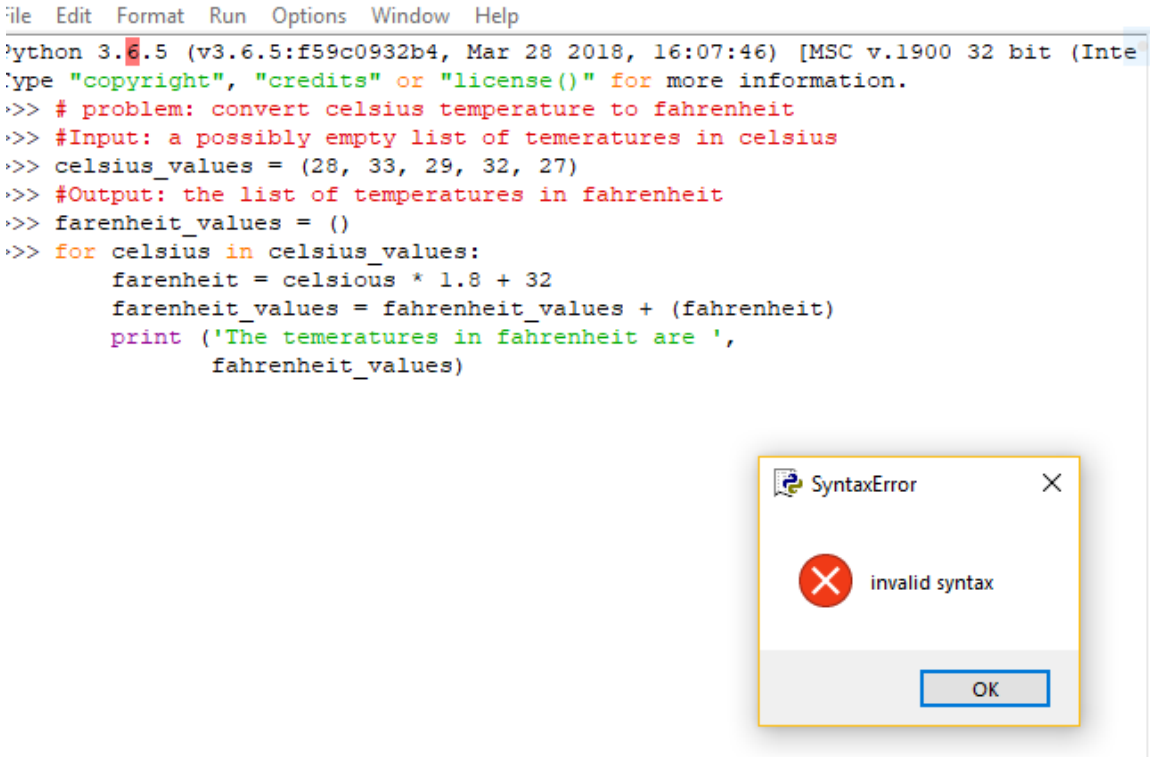


Fig. 11. Syntax error due to saving code in IDLE's shell and trying to run it in the editor

Another student experiencing similar difficulties was able to resolve it themselves:

> thanks [Anon.]
>
> I realized that the problem consisted in
> using the shell instead of opening a file
> and saving it. Therefore, having opened
> and saved the file, I could use the "Run"
> menu and let the turtle program draw the
> line. The screenshot of the file is enclosed
> below
>
> [Anon.]

5.1.2 IDE: Using the quiz framework. A separate IDE, CodeRunner [7], was used for the modules formative quizzes which involved programming questions. An issue here was students wanting to test their answers in IDLE prior to submitting them in the quiz environment - a reasonable approach since the quiz penalises incorrect answers. However, they hadn't appreciated that the quiz questions may have underpinning supporting code as pointed out by this student explanation:

```
You can't be using get_input() in idle since
it's not defined outside the test environment,
so I assume either you used input() or
something else that results in a string type.
Short answer is you cant mix  strings and ints
in addition, either 1+1 is 2 or "1" + "1" is "11",
if you try and mix and match python will throw a
TypeError exception because it doesn't know which
you want.

you can fix this by using int() to convert to
an int or str() to convert to string.
e.g. var = "1",  var = int(var). var is
now 1 (with type int), if you're using input()
you can use int(input()) to convert your input.
```

The `get_input()` function mentioned above was provided as part of the quiz IDE, but with its implementation hidden. For instance, the code field for one of the quiz questions was prepopulated with the following lines:

```
# initialise the input values
magnitude = get_input()
# if input values fall into the first case:
#     compute outputs according to the first case
(...)
```

The use of `get_input` diverted students from the main quiz activities by them:

- trying to understand what it does
- working out how to use it with their own code
- wondering why code which called `get_input` didn't work in the IDLE environment
- trying to use their own test data instead

All these points were addressed in the forums by more experienced students as demonstrated by :

> (...)
>
> Don't add anything to it don't take anything
> away from it, just leave it alone.
>
> as you surmised its purpose is to drag in test data
> without you having to do anything other than run
> the program when its in the quiz.
>
> (...)

Interestingly, the quiz question instructions also told the students to leave the line alone!

5.1.3 IDE: Basic application usage. Also, some students seemed to struggle with basic IDE functionality such as copy and paste, and saving files. For example:

> Hi [Anon.]
>
> Please, could you give me some advice on
> how to save Python files? I think I am
> doing this wrong (...)
>
> [Anon.]

and

> Hi all
>
> I wonder if anyone can help please.. is there an
> easy way of copying code within python?
>
> Whenever i copy code I always get a Syntaxerror
> so currently have to rewrite the code.
>
> I dont think ive missed anything in the
> book that says how we do this?

```
Im on a mac, not sure if that makes any
difference?

Thanks

[Anon.]
```

Which seem to suggest a lack of confidence in using transferrable skills learned when working with other software.

5.1.4 Collections. Collections featured as a topic in 21 discussions (12.5%) with most discussions revolving around lists. Students had trouble with indexing; confusing indices and contents; and inadvertently performing operations lists rather on the contents of lists.

5.1.5 Collections: Zero indexing. Here the student is trying to find the last number in a list but hasn't appreciated that Python's zero indexing means that the index will be one less than the length of that list.

```
How do i do this? Sorry for so many questions of
such a newbie nature, i'm trying to figure out stuff
and the book doesn't seem to cover it, so using
logic to try and figure it out but i have holes in my
programming knowledge that are just not helping me.

So i have tried making a list eg. [1,3,5,8,9]

Then i tried to pull out the last number by using
len(list). Something like this:

last_number=len(list)

Which pulls up the index of the last number on the
list. All very well and good.

But how do i actually use the number in this
position (in this case, it's number 9)?
When i do something like
```

```
last_number + list[0]

I just add together the index position with
the first item.

And doing

list[last_number] + list[0]

doesn't work either, i just get error messages.
And i can't find explanation anywhere in the module
material about this? Help... feeling in over
my head and i'm only on week 2. I don't
know what i'm doing wrong, i've spent a few hours
trying to figure this out and going over and over the
material.
```

5.1.6 Collections: Muddling up indexing and contents. This post combines the above problem with mixing up the role of indices and contents of items in a collection. Note that in the code below (shown in Figure 12), line 9 indicates that the student is wanting to record the position of the highest temperature but is actually keeping track of a temperature value (and not the one wanted at that).

```
I had difficulty with Q12 weeks ago and have
revisited several times, for some reason I
can't wrap my head around it when
everything else has been a breeze!

Here is where I have got to so far.

I'd appreciate it if anyone can share what they
have so far and the process of working it out. I feel
like I am missing something obvious!

Thanks.
```

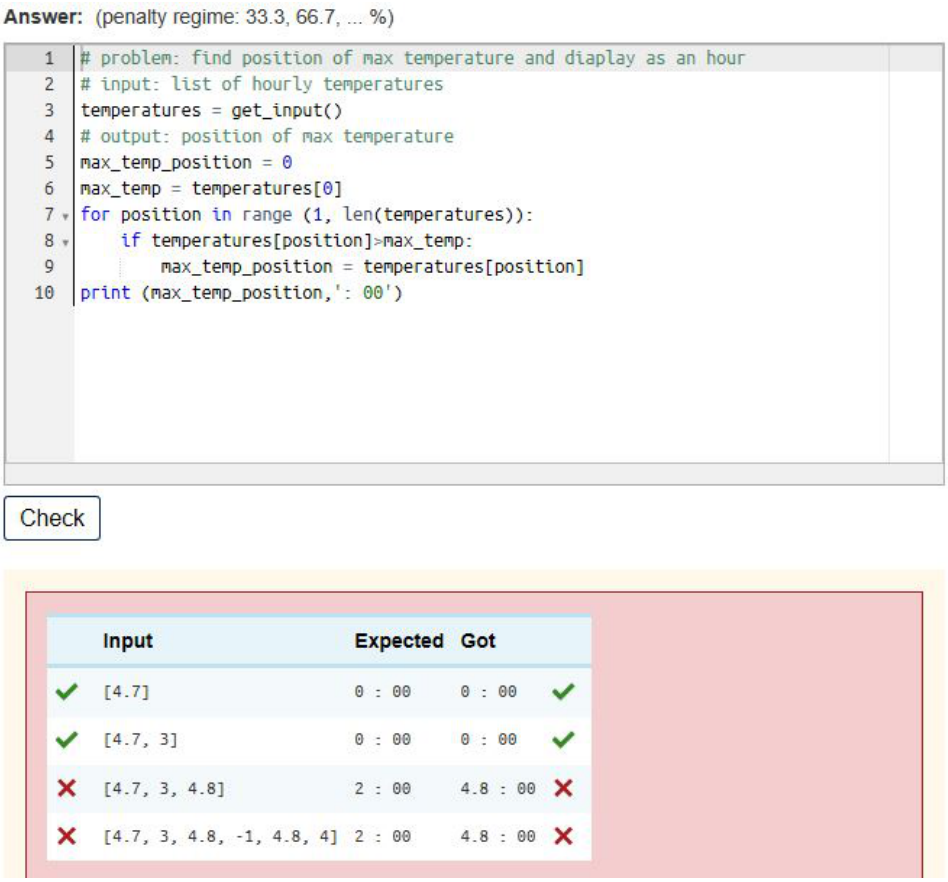


Fig. 12. Student's screen capture of quiz attempt illustrating confusion in using lists

5.1.7 Collections: Performing operations on lists rather than contents of lists. In this post the student has been trying to convert a list of temperatures in Celsius to Fahrenheit but is inappropriately using the multiplication and addition operators on the Celsius list. Although there is no evidence of students being caught out by this it is worth recalling that both multiplication and addition are valid list operators and could produce unexpected results for those unaware of them.

```
[0, 5.85, -2.5]
[273.15, 279.0, 270.65]
***Error***
Traceback (most recent call last):
  File "prog.python3", line 12, in <module>
    fahrenheit = celsius_list * 1.8 + 32
TypeError: can't multiply sequence by non-int of type 'float'
```

```
typeError: can't multiply sequence by non-int off type float!
trying to learn this one now..:P
```

```
# initialise the input_list with the given values
celsius_list = get_input ()
# initialise the output_list to the empty list
fahrenheit_values= []
negative=None
# for each input_value of the input_list:
for fahrenheit in celsius_list:
 # transform the input_value into an output_value
  fahrenheit = celsius_list * 1.8 + 32 <<<<<<< think this is the cause!
    # append the output_value to the output_list
    fahrenheit_values=fahrenheit_values+[fahrenheit]<<<<<
      # print the output_list
print( fahrenheit_values??? DO I INSERT CODE TO A OUTPUT: A TOTAL,
A NON INTEGER INTO MY PROGRAM?
```

5.1.8 Functions: Calling functions. Functions were a key theme in 16 discussions (9.5%)

Some queries were as simple as how to call a function whereas others are unsure of the syntax.

This student appreciates that the function name is needed but isn't sure about the use of brackets. Notably, the function get_input() is provided as part of the quiz environment so this may indicate that they are also struggling with the IDE.

```
Before I go ahead and test out my newely
created script that works in python.

Can I check if anyone knows if this would work?

get_input()
input_variable=get_input

the () makes me unsure
```

Whereas Figure 13 shows that this student has missed the brackets needed when calling shape()

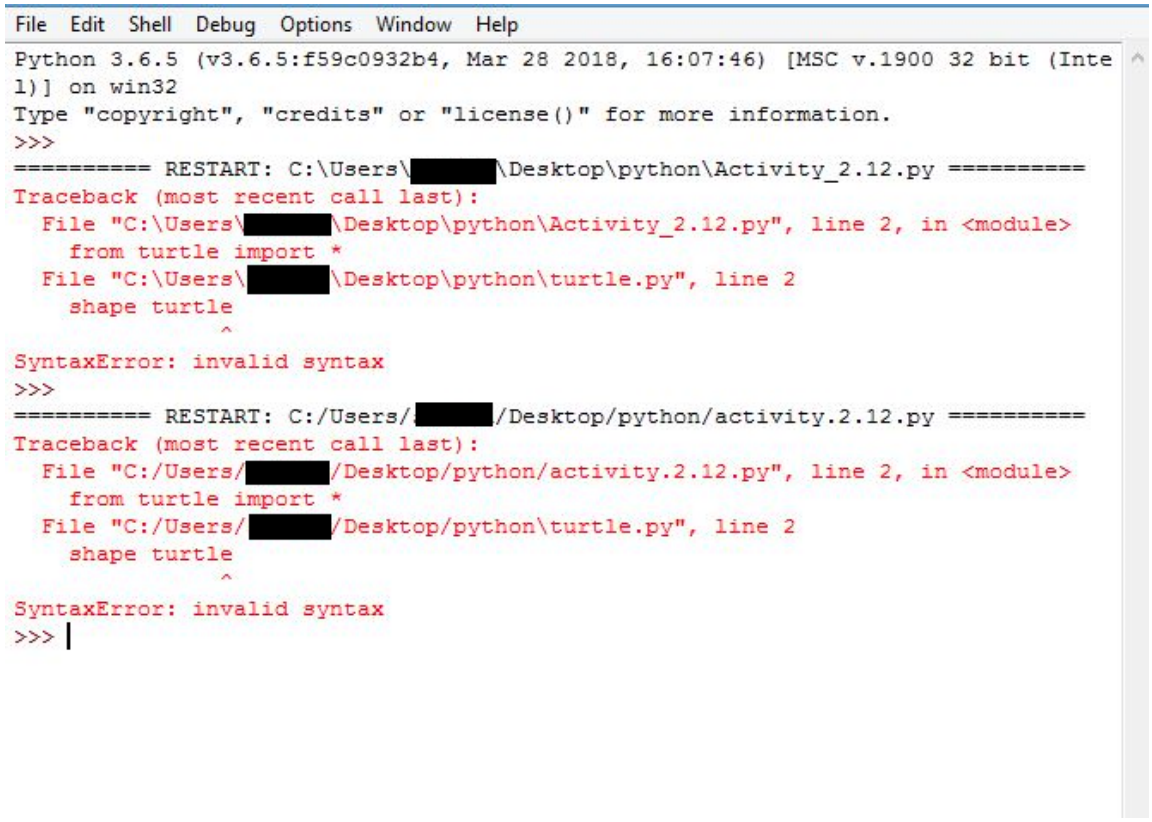


Fig. 13. Screen capture of error message when trying to invoke a method without the brackets

Figure 13: Screen capture of error message received by student when trying to invoke a method on a python object but forgetting the brackets.

5.1.9 Functions: Passing parameters to functions. In this post the student is clearly confused about passing parameters to functions and is overriding them in the body of the code:

```
Hi [Anon.]

Can you have a look at this function, Am I on the
right track???

def A_B_and_binary(A,B):
    """"""convert two inputs to tf then back to output"""""""
    A = 0
    B = 0
```

```
    truth_value(A)
    truth_value(B)
    and_binary = (A and B)
    integer_0_or_1(and_binary)
    return integer_0_or_1
```

thanks

[Anon.]

5.1.10 Error messages. Support with resolving error messages was a topic in 15 discussions (9%) with students struggling to understand what the error message was telling them. For example, the student encountering the FileNotFoundError below had not appreciated that the file "TM112_Glossary.txt" needed to be in the same directory as the Python file despite the online activity indicating this.

I tried running the following code related
to Activity 2.34 but it gave the following error
message in the shell:-

```
Traceback (most recent call last):

  File "C:\Users\User\Desktop\Flashcards_for_TM112_Glossary.py",
  line 39, in <module>

    glossary = file_to_dictionary('TM112_Glossary.txt')

  File "C:\Users\User\Desktop\Flashcards_for_TM112_Glossary.py",
  line 30, in file_to_dictionary

    file = open(filename, 'r')

FileNotFoundError: [Errno 2] No such file or directory: 'TM112_Glossary.txt'
```

This was the code taken from the module website as indicated below:-

```
(...) [several dozen lines of code omitted]
```

The problem here is more subtle in that the student had imported the entire turtle module. This module contains a function `shape()` which led to a conflict when the student tried to use a variable with the same name

```
Hi

Can anyone tell me what this type error means?

    sidelength = (0 + shape * 10)
TypeError: unsupported operand type(s) for *: 'function' and 'int'

why doesn't my variable sidelength work?

thanks

[Anon.]
```

5.1.11 Iteration. Iteration has proved to be a specific issue in about 8% (14) of discussions. A recurring theme is specifying of the range to be iterated over. The issue seems to be about establishing the correct syntax:

```
Hi,

I cannot seem to initialise for each position
from 1 to length of list - 1. I keep getting errors?

Any ideas?

Thanks
```

This is reinforced by this student who is endeavouring to understand the `for ... in ...` structure in more detail.

```
Hi,

I'm enjoying the python a lot but I really
want to understand what the commands actually
do rather than just memorise where/how to
```

> use them. Eg.
>
> for sides in range():
>
> For iteration I can understand the "for"
> and "range" bits but as for the "sides"
> and "in" bits I'm stuck.
>
> I've experimented with it a bit and
> found seemingly anything can be input
> in place of "sides" so I assume this
> is just some kind of arbitrary label to
> make the code more readable. Please let me
> know if I'm wrong on this.
>
> as for "in" I have no clue other than
> it's a syntax error to not use it. I'm
> aware "in" is a python keyword but I'm
> not sure yet what keywords are or what
> "in" really does.
>
> I'd love it if someone less of a beginner
> than I am could explain this.
>
> Thanks
>
> [Anon.]

Iterating directly through the contents of a list doesn't cause the same level of concern although the potential processing of heterogenous data did worry one student:

> I see.
>
> It's the 'iterates directly' concept that had me.
>
> I was expecting an index as I couldn't imagine
> a construction that didn't need something

> that effectively points to 'this is the 1st
> element ' , ' this is the second element ' and
> so on as it works through the list.
>
> I remember reading about python and variable
> types. It's why there is no need for initial
> type declaration. But I do note that if you
> had the list items=[1,'one'] you couldnt do
> total= items [0]+items [1] without explicitly
> casting one or the other to the correct type.
> As I recall (vaguely) in c and c++ this isn't
> an issue, but then again I seem to remember
> you can't have arrays of mixed types in those
> languages,, and a struct is a completely
> different beast.
>
> Thanks for the explanation.
>
> [Anon.]

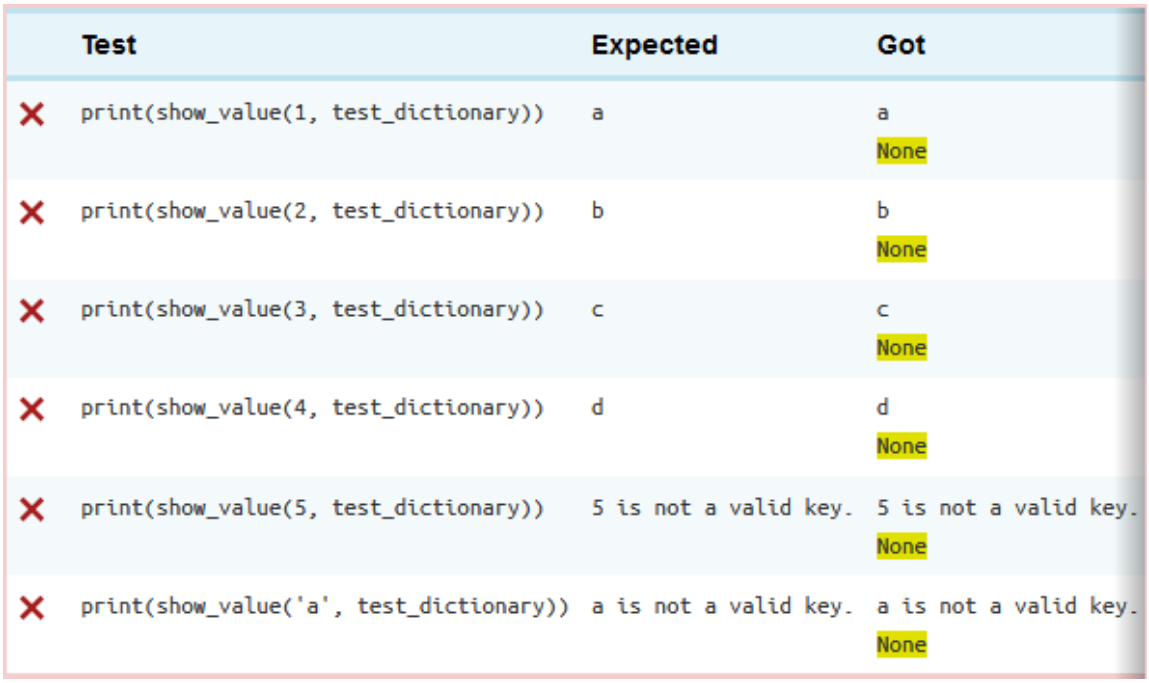
5.1.12 What constitutes output? Outputting results. 12 discussions (7%) consider output in terms of print or return and there is some confusion about the distinction between them. Students seem to appreciate that both are methods to output results from code but don't make the distinction between printing to the console or providing values to be used elsewhere in the code.

Figure 14 on Page 40 relates to a student problem with one of the quiz questions. None values are a classic sign that the code in question is not returning a value. This is later confirmed by the student after feedback from their tutor:

> Thanks for your replies. I had some feedback
> from my tutor via email and it turns out I was
> printing the value rather than returning the
> value, this is why it appeared to be the correct
> results but was still incorrect.

```
# Do not change the following line
test_dictionary = {1:'a', 2:'b', 3:'c', 4:'d'}
# Enter your function (and nothing else) below this line
```

```
def show_value(key, dict_name):
    if not key in dict_name:
        print(str(key) + ' is not a valid key.')
    else:
        print(dict_name[key])
```



	Test	Expected	Got
✗	print(show_value(1, test_dictionary))	a	a None
✗	print(show_value(2, test_dictionary))	b	b None
✗	print(show_value(3, test_dictionary))	c	c None
✗	print(show_value(4, test_dictionary))	d	d None
✗	print(show_value(5, test_dictionary))	5 is not a valid key.	5 is not a valid key. None
✗	print(show_value('a', test_dictionary))	a is not a valid key.	a is not a valid key. None

Fig. 14. Student results from online Python quiz question

A response to another activity suggests that this could be a legacy of working in the shell rather than the editor where functions can be called directly and returned values are displayed as if they have been printed. Although not mentioned in the forums, many students only previous programming experience will be based on OU Build[3] which does not have the concept of returning values.

```
Thankyou for explaining.  I thought it shoud
print the median value in the shell as in case of
mean previously.
```

[3]The Open University's adaptation of MIT's Scratch.

```
I will check again as you told.

Thankyou

[Anon.]
```

5.1.13 Indentation. Indentation was raised as an issue in 10 discussions (6%), particularly where the incorrect use of indentations was leading to bugs that students struggled to find. For example, the following code was a student's attempt at drawing four squares across the screen using the turtle.

```
# Draw squares across page
from turtle import*
number_of_shapes = 4

for shape in range(1, number_of_shapes + 1):
    # Draw a square
    for sides in range(1, 5):
        forward(40)
        right(90)
#Move forward to start of position of next square
penup()
forward(50)
pendown()
```

The student had identified that the code was indeed drawing four squares but that they were being drawn on top of each other. The solution was to indent the code responsible for moving the pen to be inside the outer loop as shown in the final four lines below.

```
# Draw squares across page
from turtle import*
number_of_shapes = 4

for shape in range(1, number_of_shapes + 1):
    # Draw a square
```

```
for sides in range(1, 5):
    forward(40)
    right(90)
#Move forward to start of position of next square
penup()
forward(50)
pendown()
```

This approach was provided by a fellow student with the following guidance:

```
I think it's cause you've not indented the penup()
part, so it's not included in  the loop. Which
means the program won't run penup() till it's
iterated over the contents of the loop first,
hence the 4 squares drawing over each other.
```

Another student exploring Python further wrote the following code, reporting that IDLE was giving indentation errors.

```
"""The echo chamber program –
This program will keep repeating
the users [sic] input back to them until the user enters 'quit'
"""
print('see␣your␣own␣input␣repeated,␣until␣you␣type␣quit.')
exit = False
while not exit:
        user_input = input('Type␣your␣input␣here:␣')
        if user_input == 'quit':
                exit = True
                else:
                    print(user_input)
```

The solution here is to outdent the final 'else:' so that it is in parity with the 'if'-line as shown below:

```
"""The echo chamber program –
This program will keep repeating
the users [sic] input back to them until the user enters 'quit'
```

```
"""
print('see your own input repeated, until you type quit.')
exit = False
while not exit:
        user_input = input('Type your input here: ')
        if user_input == 'quit':
                exit = True
        else:
            print(user_input)
```

Here, fellow students picked up on the indentation and the solution but, perhaps, muddied things through the mention of Python standards.

```
All indents must be consistent length, the
python standard is 4 spaces. In both examples
your if/else block is incorrectly indented,
the if and else keywords must be indented
the same amount, and the code blocks within
must also be the same (4 spaces more than the
if/else lines).

Edit: the indent on  the while not exit:
block is too large too, while you technically
can use any amount of spaces (as long as it's
always the same amount) you should use 4.
```

5.1.14 Variables: Naming. 8 discussions (5%) highlighted variables as an issue. Here careless variable naming (together with indentation issues) is causing a student problems. It can be seen that `temp` and `temperatures` are being used interchangeably.

```
# initialise the input_list with the given values
temperatures = [0,3,5,7,5,4,2,0,-0.2]
# initialise the output_list to the empty list
temp=[]
# for each input_value of the input_list:
    for temperatures in temperature:
# if the input_value satisfies the condition:
```

```
    if temp > 0 or temp  < 5:
# append the input_value to the output_list
    temp = temp + [temperatures]
# print the output_list
print(temp)
```

```
wondering where i am going wrong??
```

Figure 15 shows a student has fallen foul of case sensitivity in names.

Answer: (penalty regime: 33.3, 66.7, ... %)

```
1 temperatures=get_input()
2 temp=[]
3 for temp in temperatures:
4     if temp < 0 or  temp > 5:
5         temp=temp+[Temperatures]
6 print(temp)
```

Fig. 15. Variable name case sensitivity preventing code from executing

5.1.15 Variables: Declaration. These two posts illustrate some confusion about the declaring of variables in the context of loops:

```
been at this all day ..only just got on q10 ..
using module material bl2 pg 83

for guidance:

still got q10 wrong grrrr (...)
```

```
# initialise the input_list with the given values
celsius_values = get_input()
# initialise the output_list to the empty list
fahrenheit_values=[]
# for each input_value of the input_list:
for celsius in celsius_values:
    # transform the input_value into an output_value
    fahrenheit = celsius * 1.8 + 32
```

```
        # append the output_value to the output_list
        fahrenheit_values= fahrenheit_values + [fahrenheit]
    # print the output_list
    print('the temperature in fahrenheit are', fahrenheit_values)...
```

```
[Where am i going wrong]    argh!!


[Anon.]
```

It transpires that the calculation was wrong but the following post shows confusion about the variable `celsius`, clearly not understanding that it is declared in the line `for celsius in celsius_values:`

```
You've probably sorted this but I think [Anon.]
is right. The question asks for  degrees
Kelvin, so look at it again and check the
formula converting Celsius to Kelvin. I also
wondered about the word 'celsius'. Does it
need defining? You've defined celsius_value
but not celsius, does that matter?
Just a thought. [Anon.]
```

5.1.16 Imports: Purpose. Imports were also an issue in 7 discussions (4%)

Here the student doesn't grasp the purpose of the import

```
Hi [Anon.],


The code on page 98 is shown below
```

```
# Move 100 units, but ten at a time.
from turtle import *
    for section in range(10):
     forward (10)
```

```
These  4 lines of code ad me all over the place
trying to understand them,  to the extent that I
doubted what a whole number is. I looked it up
and basically it is the same as an integer, which
is a term you used in a line of your reply your
reply to Ian>>>>--range(10) generates a sequence
```

```
of integers from 0 to 9.

I understand from previous pages in the book
that we can use Python to programme a turtle,
which is fair enough.

However, line 2 of the code reads from turtle
import *   >> why is this line even in the Python
code.? Is 'turtle' some kind of computer package
we can access via  Python but only if we include
this line in the code?

[Anon.]
```

5.1.17 Imports: Usage. Whereas here the student appreciates that an import statement is needed, but is struggling with the syntax:

```
for some reason i can not open this program?

def corr_coef():        from stats_utils import

has been saved ubuntu16.04
```

Although a solution is readily available from a peer

```
Import statements must be in the format
""import <module>"" or ""from <module> import <object>""

you probably want a type of the second form
""from stat_utils import *""
```

Here the student has a valid import statement which still won't work

```
Hi all, I am on question 6,b and I am having a
problem loading the corr_coef function, I get this
error when I run my program.

from tma02_stats import corr_coefModuleNotFoundError:
No module named 'tma02_stats'
```

> Can someone check and see what's going on and how can I fix this.
>
> Thanks
>
> [Anon.]

It transpires that the the module in question is not in the same directory as the source file (or the file path for that matter):

> Hi [Anon.]
>
> I'm just guessing at the cause, but after unzipping
> the download you should be able to find a folder
> TM112_18D_TMA02_Q6-files, and inside that
>
> q6a.py
>
> q6b.py
>
> tma02_stats.py
>
> If the three files are not all together in
> all in the same folder the imports will not work.
>
> [Anon.]

The following topics all occurred in fewer than 6 discussions, see Table 2. Examples are provided but with reduced narration.

5.1.18 If-structures. Here the student has not understood the purpose of the catch-all "else" statement:

> Hi **all**,
>
> here **is** my code so far **for** this question
>
> ```
> # initialise the input values
> mag = get_input()
> # if input values fall into the first case:
> ```

```
if mag < 4:
    quake = 'minor'
# otherwise if inputs fall into the second case:
elif mag >= 4 and mag < 6:
    quake = 'moderate'
# otherwise if input falls into third case:
elif mag >= 6 and mag < 7:
    quake = 'strong'
# etc.
elif mag >= 7 and mag < 8:
    quake = 'major'
# otherwise:
else mag >= 8:
    quake = 'great'
# print the outputs

print('That␣is' ,(quake) ,'earthquake')

I am getting a syntax error on the following
line and word:

else mag >= 8:

%I dont get why....the syntax errors dont
%appear on the other mag variables...
%am i missing something obvious???

%Thanks
```

5.1.19 Software installation problems. Here the student is worried about a warning during software installation. Interestingly, it was highlighted as something that could be ignored in the instructions given to students.

```
Hi I downloaded the latest version of Python
for Mac... the one that was suggested
```

```
when I went on the website, after
downloading I clicked on IDLE but a screen
popped up saying

Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin

Type ""copyright"", ""credits"" or ""license()"" for more information.

>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.

Visit http://www.python.org/download/mac/tcltk/ for current information.

I then downloaded the version above and the
same thing happened. What to do?!

feel so stupid lol

Thanks!!
```

5.1.20 Strings. This follows on from a discussion about manipulating strings.

```
you can make the number a string then use variable = numberString + "am"

variable = numberString + "am"

numberString = str(number)
```

5.1.21 Turtle. This 'problem' with the turtle is self-explanatory.

```
When I type from turtle import * and return nothing
happens, except I get a new prompt line appear
underneath.Am I missing something?
```

5.1.22 Case sensitivity. Python is case sensitive which is a new concept for those whose only prior experience is programming in a variant of scratch in the module preceding this one.

```
I think your problem is more with using "Shape"
then "shape".
```

```
I ran the code using "shape" in the for loop and it
worked perfectly. Python (and all programming
languages) will often throw errors up pointing to the
code directly after the problem because it's expecting
something different.
```

5.1.23 Executing code. This post by a moderator aims to reassure a student who is struggling to execute their code

```
Hi
(...)
Running the provided programs won't produce any output
on its own, because all they do is load the functions
into memory.

To make any function actually execute we need to call
it, with arguments if these are required, and to display
the result.

In another post [url provided]

I've explained how we could see the result of a call
to the median() function, by running the program, then
going to the shell and calling the program.
(...)

[Anon.]
```

5.1.24 Typo. Here a student's problem is due to a typographical error, which is being pointed out to them:

```
Hi [Anon.],

Compare your use of table1 here

table1 = []
```

```
with that here

table_1.append(row)

There's a small difference.

[Anon.]
```

5.1.25 Brackets. Here a student is following up on an explanation about using brackets to control the priority of calculations but seems uncertain of the use of square brackets.

```
Sorry, i have another question... if in the above code i
wanted to do [index] multiplied by 20 instead,
would it be like this:

20 + ([index]*20)

or like this:

20 + [index * 20]

I'm thinking that it's the first one?
```

5.1.26 Character encoding.

```
I have rubber ducked myself stupid.

My code is attached.

Essentially.

This line runs ok without error

reader=csv.reader(mort_file1)

this one

reader1=csv.reader(geog_file1)
```

causes the following error message.

```
Traceback (most recent call last):
  File ""/home/[Anon.]/Dropbox/OUCloudBackupTM112/TM112Various Block 2 Files/Text and
  Python files for Week 10-20180610/activity5.32prog_17.py"", line 37, in <module>
    for row in reader1:
    File ""/usr/lib/python3.4/codecs.py"", line 319, in decode
        (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xaa in position 510: invalid
start byte
```

Any one any idea?

Is geog_2.txt faulty in some way?

Gedit is highlighting this as the issue (...)

[Anon.]

5.1.27 Code formatting. The post forms part of the communal project. Here a couple of students have been considering the approach to naming variables etc.

> My preference is underscores for readability but camelCase is more compact (although I'm sure someone will say there's one coding language at least that doesn't support underscores in names). Happily it doesn't make any difference to the code, it's just custom.

5.1.28 Docstring versus comment. Here a student is responding to a query about when to use "docstrings" as opposed to "comments"

```
Comments will never be shown unless you go into the
program code.

Docstrings are shown for functions if you call them
(ie use the help function). Any file you
save can be called in a similar way (so acts
like a function with no inputs).
```

```
Using """..."""" is also useful if you want
the program to print something over several lines
without using print() repeatedly.
```

5.1.29 File handling. Here a couple of students are analysing some code provided in support of an activity to understand how the file handling aspect of it works.

```
It uses open() to open the file with the correct
(non-default) encoding, then initialises a CSV.reader
object to read the file line by line, then returns
the contents as a list.

The "with open (...) as name:" block guarantees
that once your code is done with the file
(when the indented code block ends) the file
will close, including in the case of exceptions.

[row for row in csv.reader(infile)] is list
comprehension and it's really neat, it makes a new
list the same as:

L = []

For row in csv.reader(infile):

    L = L + [row]
```

5.1.30 File naming (.py). The post below is an explanation to a student who gave their Python file the same name as a module which led to scope problems.

```
(...)
It would cause a problem, because I think Python
would look at the 'nearest' turtle module and
that would be the program we have just written
ourselves and names turtle.py. Because it doesn't
include a definition of forward() [or any other
turtle command for that matter] we'd get the
```

```
'forward not defined error'.
(...)
```

5.1.31 Memory leaks. This post is in response to a student who had read something on Wikipedia and needed their schema amending in relation to Python and memory leaks.

> Anyone who is bemused by this thread will be relieved to learn that memory leaks are not an issue in Python, and won't be in Java, which many of you will do as your next language (but they would be for example in C).

5.1.32 Random numbers. Part of the project part (implementing a simple flashcards program that makes use of the module glossary) led to a discussion about random numbers. Here a student highlights the dangers of assuming that random number generators are truly random.

```
As a small note for real world use, you should
be very careful which random function you use for
anything that someone may be interested in
exploiting. Most functions in the random module are
actually predictable given enough time and
effort, python 3.6+ offers a module called secrets
which uses the most secure random numbers
available from your operating system.

Obviously not actually important in our little
test case, but if this were used in an online
casino or some such then a small mistake like
that could end up costing a lot of money.
```

5.1.33 Reading keyboard. Here the student is looking for alternative ways to read the keyboard.

```
I'm trying to find a way to get typed user input
without printing a newline to the terminal after.

The goal here is that code similar to
"input("some prompt"); print("\r............") will
completely overwrite the input prompt and whatever
input was typed but since input always ends with a new line
this clearly does not work.

Anyone any ideas?
```

5.1.34 Strings versus numbers. This post is clarifying a fellow students' understanding of breaking a list into individual member elements. In particular whether those elements will be strings or numbers:

```
the " or ' around an item denotes that it's a string.
When "1" is turned into an integer it becomes simply 1.
```

5.1.35 Whitespace. This post is in response to a student querying the use of whitespace in a Python statement.

```
The gap isn't important.

"Whitespace

Generally significant only to the left of code,
where indentation is used to group blocks. Blank
lines and spaces are otherwise ignored and optional
except as token separators and within string
constants."

Mark Lutz, Python pocket reference (5th Edition), O'Reilly Feb2014 Pg 69.

But having them can assist in readability.

[Anon.]
```

5.1.36 Wildcard. Here a student has a question about the use of wildcards in data-analysis.

```
Hey [Anon.],

It was the wildcard part that threw me. Don't remember
that being mentioned before. Can you use more than
one wildcard? Like filter _by(..2, 0, table1)?

Thanks for the reply
```

The respondent highlights regex which may be somewhat daunting for someone new to the language.

```
I believe filter by uses regex for its expressions,
so yes you can match pretty much any pattern
```

> you want but it might get complicated.
>
> https://en.wikipedia.org/wiki/Regular_expression

5.1.37 Windows versus Linux. This plenary post is from a student experiencing problems processing data in Linux but not Windows.

> Thank you all for your input.
>
> I have used IDLE in windows to run the activity and lo and
> behold it has no issue with Parsing geog2.txt, no errors there.
>
> My working hypothesis is that there is an issue with my
> Linux Locale settings, which are according to Gedit
> (UTF 8). This is odd because as i understand it
> there is support in UTF8 for over a million characters!
>
> Will finish today's study, then get out the reference
> books to work out how to alter my Linux locale to fix this (...)
>
> [Anon.]

5.2 Problem-solving/generic topics

Problem-solving/generic topics are those which occur across multiple programming languages and typically have the same details. For example, a typical problem-solving workflow will be largely independent of the target programming language. Similarly, discussions about how to learn Python could equally apply to C, Z80-assembly or FORTRAN.

5.2.1 Code fragment with problem. Code fragment problems were the most prevalent in terms of the number of discussions relating to problem-solving/generic topics. Typically, these posts initiated a discussion where students knew they needed support, but in many cases the information provided was insufficient to receive targeted help and indicated that they may not have the language, confidence, or knowledge to seek that support.

5.2.2 Bug finding. Over 43% of discussions (73) presented readers with a code fragment and a problem. Of these 21 related to specifically to bug finding. For example, in the 'Another quiz Q17 query!' discussion we have:

> Just catching up with the quiz, but can't see

> where I'm going wrong; am just getting the moderate
> earthquake response >= line 6 (haha.)
>
> Any direction would be great.
>
> Thanks
>
> [Anon.]

In this instance, the student did include a screen shot of part of their code and another one with the CodeRunner feedback.

And in the "where am I going wrong" discussion

> hello all would be grateful for advice i am practising
> python anf from the book have just typed in program 2:4
> but when running programme it gives me error code
> saying traceback but have followed completely to process
> so should be no problems please see diagram

(Note that the diagram wasn't provided despite repeated requests for it.)

In both these cases there is some confusion about which activity/quiz the student is referring too (although it can be deduced with some effort). Omitting the code and not communicating their efforts at debugging leads to generic advice from peers.

One of the quiz questions asks students to write code which analyses values stored in a list and to print the percentage which meet given criteria. The student's code returns incorrect results and a syntax error and they don't explain their efforts to resolve it. Guidance is offered in terms of checking variable states with print statements and the student resolves the problem themselves (an incorrect indent). Again, the guidance can be viewed as generic.

In another quiz question a different student provides much more detail including enough information to easily find the relevant material. The student has explained what they are observing and has tried to identify where the problem is. This quickly led to the problem being identified and targeted support offered by their peers.

5.2.3 Learning Python. Learning Python was raised as an issue in 15 discussions (9%). Whilst some of these were students venting frustration as illustrated by:

> (...) Just unable to get my head around Python :) Getting
> to end of my last nerve :(Sorry just had to vent.
> Hope everyone else has cracked it.

Others are more sanguine and recognize that time and commitment are often needed to master the concepts as shown here:

> (...) Sometimes it takes a while to get your head round things...
>
> I worked hard and used extra resources (fiddled around in my own time over summer holidays - and did parts of courses from places like codecademy) and the next coding module I did I understood and got on well with.
>
> The only thing with this was that I did have to take the time to push through. (...)

and here:

> (...) Understanding other people's code is a skill that will come with practice (...)

5.2.4 Maths. 6% (10) of discussions identified mathematics as an issue. Dominant amongst these were queries where student knew what they wanted to do but not how to achieve it with Python, indicating, perhaps, that the Maths topic is on the cusp of both the Python and general programming themes. For example, here a student is trying to use integer division to calculate the modulus rather than directly using the modulus operator.

> (...)
> First of all, I couldn't work out why I was getting odd results for the modulus operation. After a bit of testing and research I realised (d'oh!) that it was because of integer division. I couldn't find a way round this except by defining the constants as floats from the outset, and then simply casting them as ints for the final output
> (...)

Similarly, several discussions arose about how to return integer values when Python calculations were producing floats such as shown here:

> The issue was the format of the output integer percentage in the answer that was expected by the quiz.

```
"75" was acceptable "75.0" was not.

Rather than create a second list which was not to be
used for anything other than to be fed to the function
len() as the formal parameter (I've just read 4.1),
I counted the number of out of range items into a
variable. Even so the division part of the percentage
calculation was the issue, floor or otherwise.

Unless your method of calculation the percentage
was based on a formula something other than

(number_out_of_range / number_of_items_in_the_list) * 100

there seems to be no way of avoiding a result
that looks like 75.0.  For example.

>>> (2 / 3) *100

66.66666666666666

>>>

Would have given an incorrect answer to the quiz
question, and

>>> ((2/3)*100)//1

66.0

>>>

which rounds down, would also fail the quiz because
of the way the integer result is presented.

(...)
```

Whilst sometimes heated due to students expressing their frustration these discussions do generate a detailed discussion about how Python handles mathematics leading to creative solutions. This does indicate that the students are learning from the process.

5.2.5 Code review. Code reviews featured in 7 (4%) discussions and were typically in response to requests for help resolving a bug or to make solutions better in some way. Posts tended to give more detail than simply resolving the immediate issue. For example, this response was provided to the request from a student looking to make their code more efficient

```
Two quick comments:

- The variables new_list and old_list are misnamed
because they aren't lists, they are integers.
Naming is very important, especially as programs
get bigger. Using bad names will only confuse
others and yourself if you have to look at your
own programs much later.

- Efficiency is not the only criterion. As you
will learn later, computations can be packaged into
functions so that they can be reused. And functions
should only compute one thing,  so that they
can be reused in many contexts. So, although going
twice through a list may seem a waste, if each
of the passes through the lists does something
different, it's easier to package each of the passes
into a different function so that they can be
reused for other problems.
```

5.2.6 Problem-solving workflow. Six discussions (3.5%) incorporated posts with 'problem-solving workflow' as a topic. These were given in response to vague pleas for help where there is not enough detail to provide targeted guidance. For example this student has previously posted some code intended to process data in a list but not given any details relating to errors or problems. Peers begin with a partial code review but the student then posts this:

```
(...)
what ever i have tried it seems to not work?
where am i going wrong?.
```

```
(...)
```

And receives the problem-solving workflow guidance of

```
When ever I get to this point, I rubber duck it.

Then if I am still stuck I write code one
line at a time and check each line works.

When it does.  Next line of code.

Repeat until finished.
```

The following topics all occurred in fewer than 6 discussions, see Table 3. Examples are provided but with reduced narration.

5.2.7 Code explanation. This explanation is provided in response to a student who has attempted an activity but can't see where they have gone wrong.

```
Line 3: "and_binary = A and B" does nothing, you
never check the value of this variable untill
after its been replaced.

Line 3: if(...) The most obvious thing is
comparing against true is not needed,
"if(a==1 and b==1) == True" is exactly the same
as "if(a==1 and b==1)"

Also notable I'll advised habit that a lot of people
are doing: don't name two things the same,
naming functions and variables the same is a bad
idea. (And_binary is both in this case)
```

5.2.8 Following instructions. One post starts a discussion about following the instruction of sticking to the programming instructions in summative assessment. Students may miss the point that the assessment is to check the understanding and application of the material and that the instructions are in place to aid with that.

5.2.9 Patterns. This post extract is in response to a student who is confused by the difference between patterns, algorithms and programs:

> I think we need to capture the idea that patterns are resuable outlines; they are not algorithms for sub-problems, but problem shapes we can recognise as # ones we have met before. So we can take the previous solution and just plug in the change.

5.2.10 Algorithm. This post is in response to queries about how to create an algorithm to draw triangles. They limit their discussion due to it relating to summative assessment.

> I find that doing it methodically by hand, then for anything that's wrong in the resulting programme using trial-and-error, usually works fairly well.The starting point of the triangles is very important. Anything else I say could be seen as cheating so I'm going to steer clear.

5.2.11 How to start. This post extract is from a student who is unsure about how to approach the coding element of a question.

> Just wondering if anyone can point me in the right direction to help solve Question 5d? I'm not really sure where to look in the book to even start with this one.

Interestingly, it seems that they have not associated it with a previous question where they were asked to write the algorithm for that code as identified by this student:

> You should have written the algorithm in 5c. 5d is simply writing this out in python (...)

5.2.12 Admissible values. This student is unsure about what admissible values are.

> Hi again. This is a question I keep struggling with. What is an admissible value. I know it's part of the TMA but this is not a reference to that more just a general question. I failed with epic proportion in the first TMA when it was speaking about admissible values (literally got zilch) so I want to learn what is meant by an admissible value before starting on TMA02 Question 3.

```
Again, no ref to the question please. just a
generalisation + examples if possible - but nothing
to do with the assessment.
```

5.2.13 Barriers to learning. Lack of time to practice is certainly a barrier to learning for this student:

```
Cheers [Anon.], I'l get there in the end.....I'm not one
for giving up! I've been struggling with coding,
mainly due to lack of time to practice, I'm doing
two modules side by side, TM129 being the other.

Also working full time on shift work and I get
quite frustrated sometimes, not having time for
it to sink in. My mind does work in a logical way
I think but if it weren't for you guys, especially
yourself and [Anon.] I may have given the white flag
of surrender weeks ago!

Keep up the good work and thank you.

[Anon.]
```

5.2.14 Borderline tests. This post explains the benefits of borderline testing in response to a student's query about how we undertake testing.

```
Also when testing programs we would use data that's
well below a bordeline, just below, on the borderline,
just above it and well above it. That way you have a
good chance of finding errors in your logic.
```

5.2.15 Converting algorithm to code. This student is asking for help in converting their algorithm to Python. All the responses focused on the code that other students had produced rather than how they approached it.

```
Hi everyone,

Compute medians can use the following algorithm:
```

```
initiate the list

sort number of list from smalest to greatest value

find length of list

if length is ood:

    set median to the middle

else:

    find two middle numbers

    set median to mean of two middle number.

Could anyone help me to change the algorithm (...)
into python code? Thank you
```

5.2.16 How to explain things. This reflective post is in response to another student's appreciation of them taking the time to explain a concept in some detail.

```
No bother, I love the python problem-solving bits,
they're my favourite parts of the module (:

Putting things into words, writing explanations about
things is the part of the module I find the most difficult.
It takes me ages to write something down.  I struggle a bit
with putting sentences together and comprehension, so
I spend long amounts of time working on those parts.
I get there in the end though, after a shed load of editing.
I'm appreciating those parts now, even though I don't like
them much, cause I can see why good academic english and
making something readable is a useful skill. I think
replying to posts on the forums and trying to put something
into words is good practise for me when it comes to this,
even if I do make some
```

```
daft posts sometimes (:
```

5.2.17 Sorting. Here the student has been working with some code that sorts a list by levels of happiness. Their statement is correct demonstrating a reasonable understanding of list manipulation as well.

```
If it's sorted such that table4[0] is the happiest (or
least happy) I can't see how table4[-1] wouldn't be the
opposite, unless there's a tie for least happy?
```

5.2.18 Testing. This student extols the virtues of preventing code execution through the use of comments as a tool for testing.

```
Something I found really useful doing this question
and indeed all my Python so far is using
the 'comment out region' and 'uncomment region'
functions (under the format menu)

So you can then test each section (or subsection) of
code without re writing all the time.

As others have said use the examples in the course
material and it isn't too great a leap.
```

5.2.19 Understanding requirements. This post extract indicates that the student is struggling to understand the requirements of a quiz question.

```
Can someone please explain what it is I'm supposed to
be doing for question 9 in the quiz, without giving me
the answer? Either I'm being a bit slow today or the
question is poorly worded. Am I supposed to be creating
a list of my own to solve it, or is the answer somehow
related to that unexplained top/second line...?
```

5.3 Module-specific topics

Module-specific topics are those whose focus is particularly related to the module and have little relevance outside of it (except, perhaps, where advice may apply to other modules). For example, the module utilises online quizzes for formative assessment and discussion of those quizzes would have little relevance to the wider Python learning community. Below we discuss module-specific topics that were identified.

5.3.1 Approach to quiz, TMA and approach to activities. The three most common module-specific topics (excluding the irrelevant topic) all relate to how to tackle an item of work; either a quiz question; a TMA [4] question; or an activity from the module materials. Typically, a student has asked how to approach something or where they have gone wrong.

Responses tend to fall into three categories:

- empathy,
- a demonstration of how to achieve a goal and
- more detailed guidance.

Empathy. Empathy can be seen here in relation to a TMA question:

```
I feel your pain on this!

I thought I had a decent understanding of the basics of
Python after going through the book, but spent
close to 4 hours on this question for the TMA and
I am no closer to solving it.

I eventually had to walk away since I was ready
to throw my laptop out the window.

Hoping that a break from it will somehow give me some
additional inspiration for it.
```

A demonstration of how to achieve a goal. A demonstration, often presented 'as is' with little narration or explanation other than embedded comments, can be seen in this in response to a student struggling with a quiz question and providing the results as integer values:

```
Snap [Anon.]!
```

```
# initialise the inputs
seed = get_input()
print(seed)
# set value to the first value of the sequence
a = 9
b = 5
```

[4] A TMA, or Tutor Marked Assignment, is student course work which is formally marked by a student's tutor as part of a module's summative assessment strategy.

```
c = 8
random_num = ((seed * a) + b) % c
# print value
print(random_num)
# while the termination condition is not true:
#       set value to the next value of the sequence
#       print value
while random_num != seed:
    random_num = ((random_num * a) + b) % c
    print(random_num)
```

More detailed guidance. An example of more detailed guidance on how to achieve the task is illustrated by this response to a student seeking help with their approach to a module activity creating "flashcards" in Python (...)

```
(...)
The def show_flashcard: has completely thrown
me off track. Does the code on p.106 get joined with
other previous code related to glossary dictionary
list. Show_flashcard() from Activity 2.29 does
not work for me as indicated by the discussion.

  Yes, it needs to be added. See the first
  paragraph of Activity 2.28 on Page 105 for details.

Activity 2.31 states 'You can find a complete
version of the program on the online resource page for
this part.' I only found the python file related to TMA3.

  See the online activities section for Week 15
  at: URL provided
(...)
```

5.3.2 Forum use. Posts relating to ‘forum use’ don’t relate to students’ learning of Python but are included for completeness since they were relevant to the forum itself. For example, here a student is being reminded of forum etiquette:

> And I now realise you have already posted exactly the same
> question in here as well.
>
> Please don't cross post like this - your message will be
> read and responded to wherever you post it,
> but if you post it everywhere, you make unnecessary
> work for the mods and your fellow students may
> not see what someone else has said in another reply
> elsewhere.
> (...)

The following topics all occurred in fewer than 6 discussions, see Table 4. Examples are provided but with reduced narration.

5.3.3 General conversation. As the topic suggests some posts were broader than Python:

> PS
>
> BE BACK TOMORROW..ITS NETFLIX AND UNWIND BEEN
> STRESSFUL DAY have a great weekend all
>
> regards
>
> [Anon.]

5.3.4 Errata. This post from a moderator acknowledges an erratum and also the benefits of discovering it!

> Hi [Anon.]
>
> I never said the text is right, I was explaining
> it was incorrect! My sincere apologies for any confusion!
> There will be an erratum, obviously not until after Easter now.
>
> But the discussion has been quite useful I think,
> because it's brought out a lot of issues about
> iteration; where it start and ends; and how
> different languages do it. And the error in the text
> shows it is easy to get it wrong.

> Hope that makes sense!
>
> [Anon.]

5.3.5 Activities. This student's reflection that there is more than one way to approach programming activities is pertinent:

> ive only just come across this one, i have done it but
> the code i used to get to the next triangle is
> different to the one at the back in the answers, i
> guess its the same as OU Build [An adaptation of MIT's Scratch] where theres many
> different ways of achieving the same result...

5.3.6 Approach to studying. This student relays their approach to studying based on prior experience:

> Algorithms weren't a problem. Matrices, imaginary
> numbers, la place transforms and some other maths I
> needed for my first degree were. I nearly failed a
> couple of modules because of it.
>
> I spent a summer practising them almost every day and
> in final year used them without a problem.
>
> Getting through a module it's important to focus on
> getting as many marks as possible, but it doesn't mean
> you can't revisit things later and put in the work to
> understand them better and move forward.
>
> As an aside on hard work vs natural ability: I got the
> same overall marks as someone who did half the work but
> was naturally talented (went on to do PhD and wrote
> music albums at the same time). I literally worked
> every waking hour (and didn't sleep much) but the
> result was the same.

5.3.7 Understanding activity. Here a student is querying a peer's point about limiting code constructs that can be used in solutions. This was also an issue highlighted in section 5.2.8.

> '..I also think there is a problem with some quiz questions requiring the use of code which is either suboptimal or just pointless...'
>
> Why do you think it's suboptimal or pointless?

5.3.8 Module choices (for degree). This post extract highlights the importance of using the university's support structures for making module choices, particularly since the student in question seems to be struggling with problem solving and programming.

> So if you avoid the problem-solving techniques and programming in TM112 you will almost certainly be less prepared for some of the pathways in Computing at Level 2. So I would suggest that before you make this decision, you make sure that you discuss what your options are with the SST [Open University Student Support Team] if you decide that there are some areas of TM112 that you are finding particularly difficult. What the SST should do under these circumstances is to forward on your query to a member of the Computing and Communications department, who will know the implications for future study choices.

5.3.9 Module progress. This post extract aims to calm some students down who are alarmed that their peers are making a significant head-start with their studies before the module officially starts.

> Before anyone starts panicking at the complexity of this don't forget the module hasn't even started yet and we don't expect any of you to be able to do this right now.

5.3.10 Other modules. This student post extract responds to a peer's query about the contents of another module. Interestingly, the details about the robot programming are somewhat inaccurate.

> There's some robot programming (more similar to scratch with a less pretty interface) but a lot more discussion on what makes a robot a robot and AI and ethics, etc. Block 2 is all about networking (using a Microsoft text book with extra text telling us how to do it on other OS's). Block 3 is about OS's and it uses Ubuntu on a

```
virtual machine.
```

5.3.11 Quiz contents. This student joins the discussion about quiz contents, particularly enjoying the Python questions.

```
I really enjoy the python questions still, this module
is the first time I've used python so they're
interesting little brain teasers. I did think the
question was going to be harder than it was, but it
definitely introduces dictionaries in an easily
understandable way. My current hope is that since block
3 part 2 is called "my python project" there will be
something fun in that, and cryptography is really
interesting, so block three looks pretty good
```

5.3.12 Welcome to forum. Included for completeness, this post welcomes the cohort to the Python help forum and establishes ground rules.

```
Welcome to the 'Python help' forum.

The purpose of this forum is to

- help with any technical difficulties you have
getting Python up and running

- help you pick up the Python needed for TM112.

Please ask your Python questions and we will
do our best to answer them. If you know the answer
to a question asked by a fellow
student please feel free to post it here!

Note
 this forum is only aimed at the Python taught
 and used in TM112. If you are interested in
 discussing Python topics that range beyond the
 scope of TM112 there is a separate forum for
 this, 'Python beyond TM112'.
```

```
We hope you have a fantastic time studying the module!

[Anon.] and [Anon.], your 'Python help' moderators"
```

5.4 Miscellaneous

In addition to the topics that could be grouped with one of our three themes, we identified two further topics that fit with none of the themes: the emergent behaviour of peer support and a student instigated communal project which generated significant engagement. We discuss these two topics briefly in this section.

5.4.1 Peer support. 87% (1221 out of 1401) posts were made by students with many (about 300) of these being requests for help or direct questions. A cohort of actively engaged students answered these requests with detailed explanations and suggestions. A core group of 3 students became prominent with a further half-dozen collaborating to a lesser extent.

These supporting students were self-selecting and accepted by the community despite not being subject specialists. This is a supporting student in response to a summative assessment related question:

```
You need to work out what ""range"" means mathematically.

We really aren't allowed to give you any more help than that.

I suggest moderators delete the code and lock this post.
```

As is this, but from a different supporting student

```
Since this is a summative assessment question we
can't really help much, the answer to how to do
subtraction is fairly simple and intuitive so you
should be able to find the answer it you look.

Other than that you just have to ask your tutor
for more guidance.
```

This response is provided from a typical student. Point three relates to one of the supporting students.

```
1) You cannot discuss summative assessment
questions on this forum.
```

> 2) The mods will delete the code from your post
> because this is a summative assessment question.
>
> 3) Take [Anon.] advice.

Which elicits this from another typical student

> Especially rule number three.
>
> [Anon.] is the expert.

The supporting student in question rebukes the idea of expertise but continues to provide support in response to many other questions

> Good god no. I just read things and I'm an
> [job title provided]. Written details stand out (and I
> apparently spend too much time on this forum).

5.4.2 Communal project/Case study. One student had been experimenting with Python and shared their attempts at simulating the game of poker - to the extent of dealing random hands to a range of players - with the invitation to suggest ways to simplify the code. Although this was the only discussion of its type on the *Python help forum*, it proved significantly engaging and resulted in 32 posts (29 from students contributors and 90 readers) as the program evolved and students tried to apply a number of the principles they had learned such as manipulating lists; the difference between functions and methods; and variable naming.

There were other similar discussions on the *Python beyond TM112* forum, and this posting should have been made there. Nevertheless, we include here the following post extracts to illustrate one positive effect of such an activity:

> (...)
> It is beginning to make some sense to me. I
> wonder if TM112 is going to go into this stuff,
> i.e. Classes, Methods etc. I hope so as I
> am a noob when it comes to OOP. More like WOOPS
> when I start trying to teach myself
> (...)

and

> I have spent a worthwhile hour going through
> this implementation, understanding what is going on

```
where.

Once i had worked out that 'players' and
'player' were two different things it made it easier!!

It still wasn't clear what was going on so
i commented out the shuffling, and reduced the number
of 'players' to 4.

It then made sense.
(...)
```

6 DISCUSSION AND RECOMMENDATIONS

This report describes findings based on analysing the forum discussions of a large cohort of first-year undergraduate computing and information technology students at the UK's Open University. This is the first encounter of these students in their computing and information technology studies with the Python programming language. The report provides statistics on the topics and themes that are recurrent in the discussions. Additionally, we have included a significant sample of actual extracts from the discussions to illustrate the content and topics of the discussions.

When we compare our findings with those in the literature on the challenges, threshold concepts and misconceptions of students learning to program, we find a significant degree of agreement. For instance, even though [3] focuses on misconceptions relating to Java, there is a definite similarity with our results (e.g. their Table 4 includes topics such as returning values, calling functions, iteration, conditionals and maths which match with ones found in our study).

Many of our Python specific topics seem to be under 'Basic Programming Principles' in Table 1 of the literature review on threshold concepts by [16]. In our Python-specific top-ten, functions and outputting results resonate with the identification of function-related threshold concepts identified in [6] .

At the thematic level, [13]'s syntactic and conceptual levels (variables, conditional expressions, loops, etc.) correspond with our Python-specific topics. Their strategic level corresponds to our Problem-solving/generic topic level.

Despite this significant overlap between our findings and those in the literature, our top-ranked items are not in any of the results reported previously. Of course, our study has certain limitation and a specific scope (as described in Section 3). Nevertheless, some preliminary implications and recommendations can be drawn relating to these top-ranked challenges.

Firstly, we would like to highlight that conceptual issues can also emerge in relation to tool use (such as IDEs). In TM112, the focus was mostly on the how, rather than the underlying conceptual understanding of code execution, especially early on. This has been amended in a subsequent presentation of TM112 with informal observations suggesting a positive effect. A note of caution also emerges from our findings in relation the recommendation by Qian and Lehman [13] to make more use of existing tools. Our results did suggest that students can find it challenging to cope with several code authoring tools when learning to program.

With regards to problem-solving skills, we observed that some students struggle to solicit help, suggesting they need support early on with asking questions about their own code, e.g. by explicitly providing the code they wrote, expected output, actual output (rather than a general comment saying my code doesn't work).

This theme continued in relation to module-specific issues where we observed similar behaviour relating to tackling formative (the *approach to quiz* and *approach to activities* topics) and summative assessment (the *TMA* topic). Much of this was addressed by peer support, particularly as the community of students became more confident.

Our methodology, which differs in significant ways from most existing work on challenges for beginning programmers has resulted in findings that confirm many of the misconceptions and threshold concepts that have been identified. However, interestingly the most highly ranked challenges are not found in the previous literature and suggest further investigation.

In summary, based on our findings, we propose two preliminary recommendations:

(1) Students that begin to learn to program can be supported with wide variety of tools, e.g. for editing their code, marking it and tracing its execution. Deployment of such tools should however be undertaken with care, since adoption and use of the tools themselves can interfere with the student's learning and cause them to focus on issues with the tool use rather than the computing and programming concepts they are trying to master.

(2) Students that are learning to program should be provided with support on how to express problems with their code. This could include explicit guidance on:
   - Including the code they tried to execute.
   - Describing the behaviour/output they expected from the code.
   - Describing the actual behaviour and output.

ACKNOWLEDGMENTS

This work has been completed with support from The Institute of Coding, an initiative funded by the UK Office for Students.

We would like to thank the following students, forum moderators and module team members for allowing us to quote their forum postings in this report: Alexa Hindes, Alicia Hood, Cameron Watkinson, Chris Powell, Connor Elias, Danielle Gebbie, Darren Brown, Demi-mae Hardy, Denise Arkley, Donna Clapham, Ellen Bedson, Emily Holmes, Emma Kirby, Francis Lavery, Gareth Oliver, Garry Bell, Gerald Barrass, Ian Laidler, Ian Stockbridge, James Restall, Jason Dutton, Jason Rickers, John Howden, Kate Sim, Katrina Trigari, Lindsey Court, Luke Fletcher, Mark Williams, Michael Broadhead, Michel Wermelinger, Ming-Shih Juan, Nick Parsons, Paul Piwek, Paul Stewart, Ray Newton, Richard Walker, Richie Cuthbertson, Sakhra Haroon, Sarah Elliott-Moore, Shafia Thmor, Sharon Dawes, Shezeen Salman, Stephen Lord, Steve Woods and Tom Holmes.

We would also like acknowledge the support and advice from Patricia Charlton throughout the project.

REFERENCES

[1] Jacob Lowell Bishop, Matthew A Verleger, et al. 2013. The flipped classroom: A survey of the research. In *ASEE national conference proceedings, Atlanta, GA*, Vol. 30. 1–18.

[2] Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, Kate Sanders, and Carol Zander. 2007. Threshold Concepts in Computer Science: Do They Exist and Are They Useful?. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. ACM, New York, NY, USA, 504–508. https://doi.org/10.1145/1227310.1227482

[3] Ricardo Caceffo, Pablo Frank-Bolton, Renan Souza, and Rodolfo Azevedo. 2019. Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM, New York, NY, USA, 23–29. https://doi.org/10.1145/3304221.3319771

[4] Chris Dobbyn and Frances Chetwynd. 2014. Transforming retention and progression in a new Level 1 course. Retrieved August 29, 2019 from http://www.open.ac.uk/about/teaching-and-learning/esteem/sites/www.open.ac.uk.about.teaching-and-learning.esteem/files/files/ecms/web-content/2014-05-C-Dobbyn-and-F-Chetwynd-final-report-May-14.pdf eSTEeM project Final Report.

[5] Tony Jenkins. 2002. On the Difficulty of Learning to Program. In *Proceedings of the 3rd Annual HEA Conference for the ICS Learning and Teaching Support Network*. 1–8.

[6] Maria Kallia and Sue Sentance. 2017. Computing Teachers' Perspectives on Threshold Concepts: Functions and Procedural Abstraction. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17)*. ACM, New York, NY, USA, 15–24. https://doi.org/10.1145/3137065.3137085

[7] Richard Lobb and Jenny Harlow. 2016. Coderunner: A tool for assessing computer programming skills. *ACM Inroads* 7, 1 (2016), 47–51.

[8] Davin McCall and Michael Kölling. 2014. Meaningful Categorisation of Novice Programmer Errors, In Proceedings of the 2014 IEEE Frontiers in Education (FIE) Conference. *Proceedings - Frontiers in Education Conference, FIE* 2015. https://doi.org/10.1109/FIE.2014.7044420

[9] Erik Meyer and Ray Land. 2003. Thresholds Concepts and Troublesome Knowledge: Linkages to Ways of Thinking and Practising within the Disciplines.

[10] Christian Murphy, Dan Phung, and Gail Kaiser. 2008. A distance learning approach to teaching eXtreme programming. *ACM SIGCSE Bulletin* 40, 199–203. https://doi.org/10.1145/1384271.1384325

[11] Paul Piwek and Simon Savage. 2020. Challenges with Learning to Program and Problem Solve: An Analysis of Student Online Discussions. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. ACM, New York. http://oro.open.ac.uk/68074/

[12] Paul Piwek, Michel Wermelinger, Robin Laney, and Richard Walker. 2019. Learning to Program: From Problems to Code. In *Proceedings of the 3rd Conference on Computing Education Practice (CEP '19)*. ACM, New York, NY, USA, Article 14, 4 pages. https://doi.org/10.1145/3294016.3294024

[13] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[14] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[15] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan Erik Moström, Lynda Thomas, and Carol Zander. 2012. Threshold Concepts and Threshold Skills in Computing. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research (ICER '12)*. ACM, New York, NY, USA,

23–30. https://doi.org/10.1145/2361276.2361283

[16] Kate Sanders and Robert McCartney. 2016. Threshold Concepts in Computing: Past, Present, and Future. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16)*. ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2999541.2999546

[17] Simon Savage and Paul Piwek. 2019. Full report on challenges with learning to program and problem solve: an analysis of first year undergraduate Open University distance learning students' online discussions: Python source code. (12 2019). https://doi.org/10.21954/ou.rd.11336105.v1

[18] Dermot Shinners-Kennedy and Sally A. Fincher. 2013. Identifying Threshold Concepts: From Dead End to a New Direction. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 9–18. https://doi.org/10.1145/2493394.2493396

[19] Elaine Thomas. 2019. A new approach to teaching introductory Computing and Information Technology by distance learning - addressing key issues. In *Connecting through Educational Technology - Proceedings of the European Distance and E-Learning Network 2019 Annual Conference*, Airina Volungeviciene and András Szűcs (Eds.). 292–300. http://oro.open.ac.uk/62184/